APPLICATION COORDINATION INFRASTRUCTURE FOR UBIQUITOUS COMPUTING ROOMS

A DISSERTATION

SUBMITTED TO THE DEPARTMENT OF ELECTRICAL ENGINEERING

AND THE COMMITTEE ON GRADUATE STUDIES

OF STANFORD UNIVERSITY

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

FOR THE DEGREE OF

DOCTOR OF PHILOSOPHY

Bradley Earl Johanson

December 2002

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

_____

Armando Fox, Principal Adviser

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

_____

Patrick M. Hanrahan

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

_____

Terry A. Winograd

Approved for the University Committee on Graduate Studies:

_____

THIS PAGE LEFT INTENTIONALLY BLANK.

Abstract

The common mode of interaction with a computer involves a single user using a mouse and keyboard in front of a single display. With the proliferation of mobile devices and the ability to embed large touch screens in workrooms, a new type of human computer interaction involving multiple machines and multiple people becomes possible. This research looks at the case of ubiquitous computing rooms, or interactive workspaces. These are device rich spaces where groups come together to collaborate on solving some problem. The nature of interaction in such a space will be with collections (or ensembles) of applications, and to date there is no clear model of how these applications should coordinate with one another at the application level.

This dissertation addresses the need for a well-suited middleware infrastructure for application coordination in interactive workspaces. It consists of three main contributions. The first is a characterization of interactive workspaces and the requirements for a coordination infrastructure for such space. The second is the Event Heap coordination model, which is an extended version of the tuplespace model. It consists of a set of properties, of which the most key is decoupling applications to reduce interdependency and enhance failure tolerance, and a set of features that together define a coordination infrastructure suitable to the characteristics of interactive workspaces. The final contribution is a validation of the model through the development and use of a prototype implementation of the model in many interactive workspaces and applications over a three-year period and around the world. An evaluation of the validity of the model is also provided in the form of an overview of the applications and deployments, a survey of developers and administrators, and experimental measurement of the performance of the prototype implementation.

THIS PAGE LEFT INTENTIONALLY BLANK.

Acknowledgements

While the contribution of this dissertation is my own, I wouldn't have been able to finish it without the support of many.

First, I would like to thank Armando Fox who has been my primary advisor for the past two years. Before he joined the Interactive Workspaces group I was floundering around as the only full time systems person, and a very inexperienced one at that. His guidance has allowed me to better appreciate the nuances of systems work and refine the Event Heap model and its implementation. Thank you Armando.

Pat Hanrahan also deserves my gratitude. Pat ends this dissertation as my co-advisor, but began as my primary advisor many years ago. He was willing to take me into the Graphics Group with little more than a single recommendation letter and gave me guidance as I struggled to find a project that matched my interests and skills. Once I settled on systems infrastructure for interactive workspaces, he continued to guide me until Armando arrived on the scene, even though systems was not his area of specialty. I should further note that the Interactive Workspaces project would probably never have happened without Pat. He and Terry Winograd, decided to extend their research in large displays to include rooms full of displays and devices, and, without formal funding, committed to building the first iRoom. Without this prototype interactive workspace, my research would have never gotten off the ground.

Terry Winograd, the third member of my reading committee, also deserves thanks. Beyond his help with pushing the Interactive Workspaces project forward, he has also played an inspirational role for me through the years. Watching his work and the work his students, on the Human Computer Interaction (HCI) aspects of the project inspired a lot of the functionality that found its way into the Event Heap. My one-on-one meetings with him, in particular with relationship to PointRight, have also always proved helpful in clarifying my thinking. Finally, I would like to thank Terry for his detailed reading of this dissertation and the extensive comments that he returned to me. By responding to his criticisms, this dissertation has definitely become a stronger work.

I also want to briefly thank both Fouad Tobagi and Bill Dally. Professor Tobagi was my original advisor and any desire I have to be clear and rigorous was inspired by him (my natural tendency is to cut corners and just "get things done"). Professor Dally served as a bridge advisor for a few quarters as I switched from networking to computer graphics. I have tried to apply his

methodology of keeping short-term goals and continually evaluating progress toward them to my own research, and have particularly found this helpful in times when progress was slow.

All of the papers that I have worked on have been done with co-authors, and none of them would have reached publication without their help. I'd like to thank all of my co-authors. Besides my committee members, Armando, Pat and Terry, I have worked with Shankar Ponnekanti, Emre Kıcıman, Maureen Stone, Greg Hutchins, and Caesar Sengupta.

Beyond that, I also thank the set of people that have been in the infrastructure sub-group of the Interactive Workspaces Project over the years. Aside from Shankar, Emre and Caesar, I in particular thank Susan Shepard, Penka Vassileva, Brian Luehrs, and Tico Ballagas. Bryn Forbes and Rito Trevino, who, along with Greg Hutchins, helped to build the original iRoom, are also owed a debt of gratitude. In the greater Interactive Workspaces project I thank Jan Borchers for pushing ahead the iStuff project, one of the best uses yet of the Event Heap, and for helping us to get funding through the iSpace initiative. François Guimbretière, despite his pessimism, overall was encouraging through the years and I was inspired by his impressive work on the Post-Brainstorm system. I will also always hear his voice in my head, "What about the user?" whenever I think about taking a shortcut which will just make life harder for developers or end-users down the road. Unfortunately, the Interactive Workspace project is now too large to thank everybody individually, but they deserve collective thanks.

Of course, there is a world beyond research (although sometimes it doesn't seem that way!). Through the years I have been involved with the Cornerstone Christian fellowship on campus, and want to thank Scott Dudley, Libby Vincent and the other staff members that have helped to keep me spiritually healthy through the years. I also want to thank the members of the several small-group Bible studies in which I have participated for encouraging and supporting me through the years. Thanks Derek, Huck, Blake, Conant, Matt, Fritz, Scott, Eric, David, Art, Mark, Julian, and Hollis!

Many other friends have also provided support through the years. I can't name all of them individually, but I appreciate every single one of them. In particular, I'm grateful to Todd Neller, my roommate from my first year in graduate school. He befriended me when I knew nobody at Stanford and has stood by me through the years.

Of course, I would not be here without my family. Thanks Mom, and thanks Dad! They supported me through school (academically, financially and emotionally) and encouraged me to strive for higher and higher levels of academic achievement, helping me anytime I had trouble. I

also thank my brother Curt. He was there through the good and bad parts of our childhood, and by not being like me he has challenged me to examine things I might not normally have considered. I also owe thanks to my grandparents, now deceased, who were always interested in what I was doing. I miss them all very much, in particular my Nana, Dorothy Willets, who was always an inspiration to me. My extended family of half-brothers, half-sisters, aunts, uncles and cousins are also meaningful to me and I'm grateful for having every one of them.

And finally, but most importantly, I thank God for walking beside me through every day of my life, and Jesus Christ for his sacrifice for me and for many on the Cross. I am fundamentally a broken and incapable being, but the strength of the Father, the Son and the Holy Spirit have allowed me to do and experience some remarkable things, only one of which is getting through the Ph.D. process. With God walking beside me there is never a dull moment, and I look forward to what comes next!

-Brad Johanson, November 20, 2002

THIS PAGE LEFT INTENTIONALLY BLANK.

Table of Contents

List of Tables

THIS PAGE LEFT INTENTIONALLY BLANK.

List of Illustrations

Chapter 1 – Introduction

When computers were first invented, nations had at most a few computers, and they filled large rooms. Since then, computers have evolved through the punch-card era, time-sharing systems, and PCs. While each step in this evolution did not necessarily eliminate the style of computing of the previous age, each step did require a change in both how software was designed and the human computer interface in order to reduce the human overhead for programming and running the machines. In the beginning, it was feasible to have large teams just to program and use the large computer, but this had to evolve toward smaller teams, single-skilled users, and then single casual users for the continually wider deployment of computing to continue.

As pointed out by Weiser [142] and others, we are now entering a new age in this evolution as we switch from one computer per user at any given time, to a continuous interaction of users with many different computers in the environment around them—the age of ubiquitous computing. While the technology exists to create the devices, ubiquitous computing will not become a reality until the software tools exist that allow for the straightforward development of applications that run across all these machines, and the human computer interaction techniques have been developed that allow users to interact productively with all these devices with minimal overhead.

This dissertation presents the Event Heap system model for coordination amongst applications running in one sub-domain of ubiquitous computing, ubiquitous computing rooms, or *interactive workspaces*. This introduction gives a brief background context for this work within the Interactive Workspaces project. It begins by introducing the concept of ubiquitous computing in more detail, presents information on the Interactive Workspaces project at Stanford University, and then describes the major contributions of this research. It concludes with an overview of the remainder of the dissertation. Much of the material presented in the remainder of the introduction

is drawn from [86], an overview article of the Interactive Workspaces project written by the author, Armando Fox and Terry Winograd for Pervasive Computing magazine.

1.1 Ubiquitous computing

The term *"ubiquitous computing"* was coined by Weiser in his seminal paper "The Computer for the 21st Century" [142]. According to him, ubiquitous computing will have arrived when the computers we use "weave themselves into the fabric of everyday life" such that they are no longer noticed. Computing in 1991, he wrote, was like writing when you had to make your own ink and bake your own tablets on which to write. In this respect, not much has changed in the world of computing in the past 11 years. Interacting with computers is still a very conscious act fraught with technology-specific overhead unrelated to the task being undertaken.

Ubiquitous computing remains, however, ill defined. In general, it has to do with the continuing transition from an age of computing with few machines serving many people (mainframes, mini-computers and earlier), through an age of one person per computer (the PC), to an age where each person interacts with tens, hundreds or thousands of devices in their environment. Some of these may be personal devices like cell-phones, PDAs, laptops and PCs, and some devices may be part of the environment, such as the computers built into automobiles.

Before going into more detail of what is ubiquitous computing, let us first clarify what is *not* ubiquitous computing. Looking first at Weiser's own definition, ubiquitous computing is not *multimedia*, where the main goal is to attract attention—ubiquitous computing seeks to augment the capabilities of the user without distracting him from the task at hand. Ubiquitous computing is also not *virtual reality*, whose primary goal is to allow one to escape entirely from the everyday world—ubiquitous computing seeks to be intimately integrated with everyday activity in the real world.

Other researchers [95] have also stated that ubiquitous computing requires this physical integration. They go further and state that spontaneous integration across devices is also an important hallmark of ubiquitous computing. Based on these two points, physical integration and spontaneous interaction, they identify that actions such as using your laptop at a conference, even with a network connection, are not ubiquitous computing since there is no integration with the surrounding environment, and no interaction across local devices.

The two major sub-areas of ubiquitous computing are *mobile computing*, where computational devices that you carry around allow you to interact with environments you encounter, and *environmental computing*, where you interact with computers embedded in your local

environment. The term *pervasive computing* has also been used to describe both of these types of computing, and is often used synonymously with ubiquitous computing.

Ubiquitous computing is still in its infancy as a field. While some characteristics of ubiquitous computing are coming to be accepted, much still remains to be resolved. In particular, there is no good model for developing applications that allow for the "spontaneous interaction" said to be key to ubiquitous computing. Further, there are few development tools available, and those that are available tend to be crude. In many respects, the field is in a similar position to parallel computing in the mid-1980s. At the time Ahuja, et al. [20], made a statement about parallel hardware and the inadequate means for developers to create programs for those machines. The quotation, when paraphrased, applies well to the current state of ubiquitous computing:

> "Many uses of ubiquitous computing are known, and more are likely to arise; hardware to support ubiquitous computing is available, and much more will be soon. But the fate of the whole effort will ultimately be decided by the extent to which working programmers can implement the ubiquitous computing applications that will run on these diverse collections of hardware. The needs of ubiquitous computing application developers have not been accommodated very well to date." - After Ahuja [20]

1.2 Interactive Workspaces—Technologically Enhanced Team Project Rooms

Ubiquitous computing is a broad domain, encompassing many different kinds of settings and devices, and it is still not clearly defined. In the Interactive Workspaces project at Stanford, we chose to narrow our focus by:

- Investigating how to map a single defined physical location to an underlying systems infrastructure, and a corresponding model of interaction (called the 'boundary principle' in [95]).
- Emphasizing the use of large walk-up displays, some using touch interaction.
- Collaborating with other research groups both inside and outside the Computer Science department to construct "non-toy" applications in design and engineering.

Specifically, this meant investigating technologically-augmented room environments with large embedded touch screens and support for users bringing devices with wireless connections into the space. In the broader research community, these are generically known as *ubiquitous computing rooms*.

Since the project began, we have constructed several versions of our prototype interactive workspace, called the iRoom, created a software infrastructure for this environment called

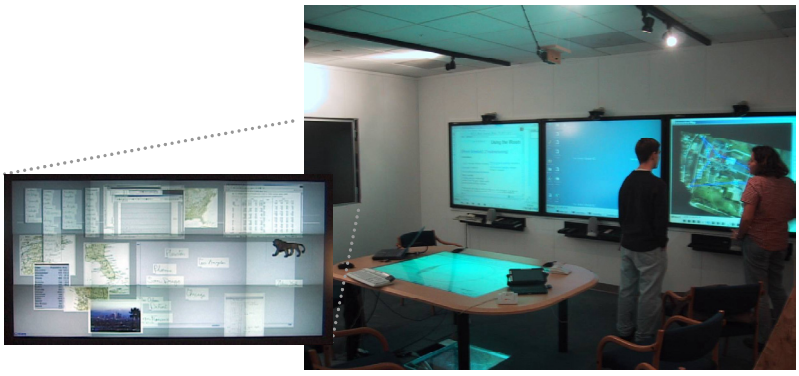
iROS—of which one piece, the Event Heap, is the topic of this dissertation—conducted experiments in human-computer interaction (HCI) in the space, and assisted outside groups in using our technology to construct application suites that address problems in their own domains.

As we began to construct our first prototype workspace, we developed some guiding principles:

- **Practice what we preach:** From the point when the dust began to settle, we have used the iRoom as our main project meeting room and have employed the software tools that we constructed. Many of the research directions and many of the tools we have built were motivated by our frustration at encountering something we could not accomplish in the iRoom.
- **Emphasize co-location:** There is a long history of research on computer supported cooperative work for distributed access (teleconferencing support). To complement this work, we chose to explore new kinds of support for team meetings in single spaces, taking advantage of the shared physical space for orientation and interaction. We were inspired to work in this space in part by work by other researchers, such as Streitz [136], which showed video conferencing and other remote collaboration tools to be inadequate substitutes for collocation.
- **Reliance on social conventions:** Many projects have attempted to make an interactive workspace "smart" (usually called an *intelligent environment*) [34, 47]. Rather than have the room figure out what a group is doing and react appropriately, we have chosen to focus on providing the affordances necessary for a group to adjust the environment as they proceed with their task. In other words, we have set our *semantic Rubicon* [95] such that users and social conventions take responsibility for actions, and the system infrastructure is responsible for providing a fluid means to execute those actions.
- **Wide applicability:** Rather than investigating systems, application suites, and their use just in our specific space, we decided to investigate software techniques that can be used in similar but differently configured interactive workspaces. Our goal is to provide a framework that serves a role similar to the device-driver and window-manager system model, and look and feel HCI guidelines for PCs. In other words, we want to create standard abstractions and application design methodologies that apply to any interactive workspace.
- **Mechanization, not-automation:** In [116], Panko makes the observation that there are two types of office environments: those that are highly structured and routine, such as

offices which process forms, and those that handle non-routine, constantly changing, information processing tasks, such as those where managers and professionals address a continuously changing business environment. Panko suggests that these latter offices are the wave of the future. Unlike the former offices where things can be *automated*, eliminating the need for human labor, the latter can only be *mechanized* by giving the workers a collection of tools they can use in pursuing their task. This latter type of worker and the tasks they perform are what we chose to address in the Interactive Workspaces project.

(a) The First Version of the iRoom



(b) Version Two of the iRoom with Inset of High-Resolution Display

Figure 1 - A View of the Interactive Room (iRoom)

1.2.1 The iRoom

As our first goal states, we felt strongly from the beginning that we wanted to not only do research in the area of interactive workspaces, but also use them in our day-to-day lives. To accomplish this, we constructed the iRoom, short for Interactive Room, which is currently in its

second generation (see Figure 1). Several other iRooms are being created both at Stanford and at elsewhere.

The original iRoom consists of three 5' diagonal SMART Board [13] touch-screens along one wall, a bottom-projected tabletop display, and a 6' diagonal custom high-resolution front screen. All but the front screen are driven by standard Windows 2000 PCs. The room also has 802.11b wireless support for laptops and hand-helds (we have also been doing some initial experiments with Bluetooth [76]). There are a number of additional devices, including cameras, a ceiling-mounted tabletop scanner system and a collection of wireless buttons.

Most of the hardware in the room is standard commercial equipment, except for the front screen, called the Information Mural, and the table, called the iTable.

The Information Mural is a 12-projector tiled back-projection display, which provides 64 dpi of resolution. Even though it is 6' across, its nine mega-pixels provide approximately the same resolution as a standard 21" monitor running at 1024x768. This allows both fine work at the surface and the ability to stand at a distance to get the overall picture. The mural is powered by a 32 PC rendering cluster and a custom software system called WireGL [81], which allows it to appear to applications as a single large OpenGL device.

The iTable was custom designed to include a 3'x4' back projected display screen in a standard conference room. A projector and mirror under the floor bring the image to the table surface. No barrier is used to prevent legs from obstructing the light path, which makes the iTable look more like a standard conference room table. Our hypothesis, which has been borne out in use, was that people would move their legs when they realized they were casting a shadow on the screen. Unlike a wall display, there is no shared "up" direction for a user interface (UI) on the table, so standard interfaces are not optimal. Karen Grant, one of the members of the HCI group on the project, has just begun to investigate omni-directional UIs for the iTable.

The choice to provide lots of display space in the room, and in particular, to have a large-high resolution display was motivated in part by the work of other researchers. For example, in a study of team project rooms [83], Johansen found that, "Teams frequently like to work at round or oval tables that allow enough space for one side of the room to function as a display wall." This configuration is how the iTable and the three SMART Boards in the iRoom are set up. For brainstorming, Applegate found [22] that "an important dimension of the consensus development process is that the group members must be able to view the entire problem and not be limited by the view of the problem provided by a computer screen." By providing lots of screen real estate,

we attempt to eliminate this problem. Finally, Covi [50] found in a different study of team project rooms that there was a need to see the big picture from a distance, but also be able to walk up close for detail. This need for both up close and distance work helped motivate us to design the Information Mural and integrate it with the iRoom.

History of the iRoom

The iRoom has gone through several iterations, which are briefly described here:

Early Work: The Information Mural

The project started with the Information Mural, a four-projector tiled display built in several stages from 1998 to 1999. It included a pressure sensitive floor, which tracked users in front of the display with 1' accuracy. The pressure sensitive floor was used in some artistic applications, but has not been duplicated in the iRoom.

iRoom v1

The first iteration of the iRoom was constructed in summer 1999. Like the current version, it had three SMART Boards and the iTable, but it had a standard front projected screen instead of the Information Mural at the front of the room.

Perhaps the biggest mistake we made in the construction of the first iRoom was in planning the cabling. It seems like an obvious thing, but the number of cables needed to connect mouse, keyboard, video, networking, USB devices etc. quickly escalates, leaving a tangle of cables that are not quite long enough or going to an unknown device. For iRoom v2, we learned from our mistakes and made a careful plan of cable routes and lengths in advance, and made sure to label both ends of every cable with what they were connecting.

iRoom v2

In summer of 2000, the Information Mural was integrated as the front display of the iRoom. This required a reconfiguration of the entire workspace. During the remodel we introduced more compact light-folding optics for the projectors on the SMART Boards, and did a better job of running the over one-half mile of cables for the room. Based on a plan arrived at with the help of designers from Ideo [4], a developer lab adjacent to the room was added, along with a sign-in area that holds mobile devices and a dedicated machine that can be used for room control. The developer station has been configured with a KVM (Keyboard-Video-Mouse) switch so that all of the iRoom PCs may be accessed from any of four developer stations. The floor plan of iRoom 2 is shown in Figure 2.

One of the big headaches in building iRoom v2 was dealing with projector alignment and color calibration for both the tiled display and the SMART Boards. Maureen Stone, one of the senior researchers on the project, has written more about the color problems in [135]. François Guimbretière presents more detail about specific mechanisms we deployed for projector alignment in his dissertation [73].

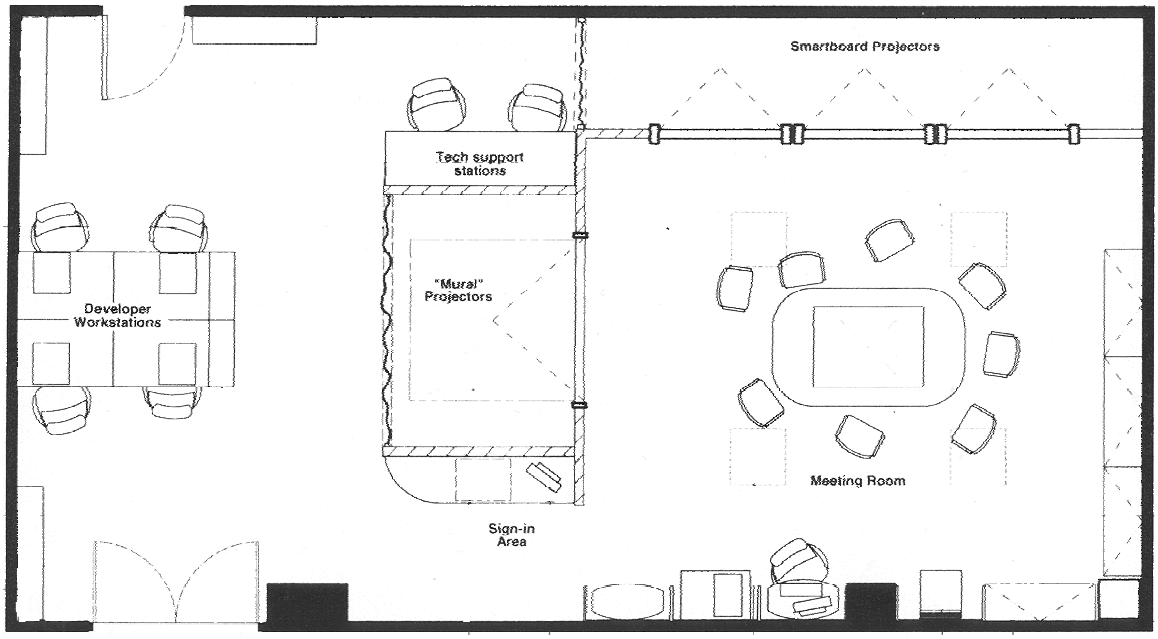


Figure 2 - Floor Plan and Behind the Scenes of iRoom v2

The Proliferation of Interactive Workspaces

Since the building of iRoom v2, Interactive Workspaces technology has been deployed at many more locations around campus, including a prototype construction trailer setup at CIFE, a future classroom prototype at the Stanford Learning Lab (now called the Stanford Center for Innovation in Learning, or SCIL) [14], a working classroom used for a writing class, and three new classrooms in Wallenberg Hall. Through the iSpace project [6], Interactive Workspaces group software is now being used in Sweden, some civil engineers in Switzerland have set up an iRoom, and several sites in Germany are experimenting with our software. The i-Land [136] group has also done some initial work to use the Event Heap in conjunction with their own software framework.

1.2.2 iROS

The Interactive Workspaces project spans several different disciplines within Computer Science, including Human Computer Interaction (HCI) and Systems research, as well as outside fields, such as Mechanical Engineering and Civil Engineering, that are investigating applying the

technology to their own domains. All of these disciplines must interact with one another for the research to be meaningful. Specifically, the HCI researchers must take into account the needs of the specific domains that will be working in the space, and the Systems researchers must produce infrastructure that provides for the HCI researchers.

This dissertation concerns the Systems aspects of Interactive Workspaces. In this sub-area of the project, we are looking at what sorts of infrastructure can support HCI that moves beyond the desktop PC model of devices, drivers, and GUIs. The need for such infrastructure has been articulated in more detail by Terry Winograd, one of the principal investigators of the Interactive Workspaces project, in [144].

The overall system we have created to address the need for post-PC infrastructure is called iROS, which stands for Interactive Room Operating System. It is a best viewed as a meta-operating system or middleware infrastructure, tying together devices within an interactive workspace that each have their own low-level operating system. It is designed so that one instance of iROS runs in each interactive workspace. To coordinate and interact with other devices and applications in the workspace, a new device or application needs to contact the iROS infrastructure running there.

The three iROS sub-systems are designed to address three key user needs in an interactive workspace: moving data, moving control, and providing for coordination between applications running in the workspace. The Data Heap provides a data storage and transportation mechanism to make it easier for applications to move and retrieve data in an interactive workspace. iCrafter makes moving control easier by providing a standard way to generate a user interface for any device. The Event Heap, which is the topic of this dissertation, is designed to facilitate dynamic application coordination, as well as supply the underlying communication infrastructure for applications and other systems infrastructure in an interactive workspace.

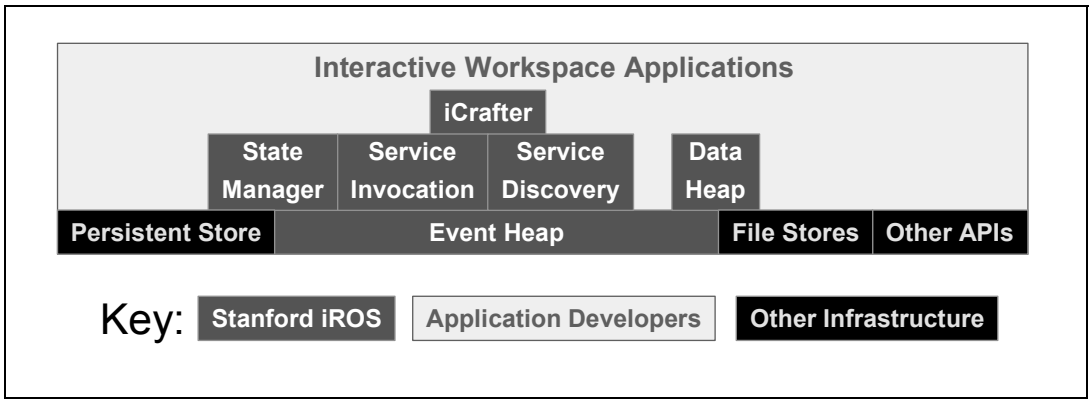


Figure 3 - iROS Component Structure

Figure 3 shows how the iROS components fit together. The only component that an iROS program must use is the Event Heap, which is the underlying coordination and communication infrastructure for applications within an interactive workspace.

1.3 An Application Coordination Infrastructure for Interactive Workspaces

This dissertation presents the motivation for why the Event Heap design is well suited to provide the coordination component of the iROS system. It consists of three major contributions to the body of knowledge:

- **C1:** A characterization of Interactive Workspaces from the standpoint of factors that must be taken into account by any coordination infrastructure in such a space. This is in the form of a set of characteristics of interactive workspaces.
- **C2:** The Event Heap, an infrastructure model for application coordination in an interactive workspace. The model has two parts. First, there is a set of properties that a coordination infrastructure needs to have to provide for the characteristics laid out in **C1**. Second, an extended tuplespace model is specified whose features supply the needed properties.
- **C3:** Instantiation of the Event Heap model as a working piece of infrastructure software, and demonstration that it serves its purpose through its use in dozens of applications running in at least ten different prototype workspaces.

While work has been done in ubiquitous computing rooms for many years now, most of the actual systems work has been *ad hoc* and designed only to support some higher-level research in the domain of technologically enhanced team project rooms. In part, therefore, our contributions can be seen as addressing three key problems:

- There was no good model of the characteristics of interactive workspaces that could inform the design of infrastructure for such environments. Contribution **C1** addresses this.
- There was no reasoned model for a coordination infrastructure for interactive workspaces. Contribution **C2** addresses this.
- There were no good, general, tools to create applications for interactive workspaces. iROS is designed to address this in general, and the Event Heap implementation, contribution **C3**, addresses the coordination-specific needs of developers.

The reader should be aware that most of the system properties and features that we propose are not being claimed as contributions, since each is present in one or more other coordination infrastructures. What is new is the analysis of what set of properties and features are necessary in the interactive workspaces domain, and the implementation of the only system that we know of that has this particular combination of properties and features.

During the past three years we have had the chance to design and implement several iterations of the coordination infrastructure that we use in the iRoom and that has become the Event Heap. Our experience with the flaws that limited the ability of the system to provide support for meetings and collaboration sessions informed the design of each new generation, and has led to the Event Heap model presented in this dissertation. Beyond specific design decisions, a principle of decoupled-design emerged through our experiences. Throughout the Event Heap model, and within other iROS systems, we have held to this principle, minimizing the amount of direct inter-component interaction wherever possible. We believe the combination of failure tolerance and interoperability we have been able to realize would not have been practical using any other mechanisms.

In his original paper on ubiquitous computing, Weiser asserted that software infrastructure and appropriate development tools were going to be critical for the field to move forward. Our hope is that with this dissertation, a step in this direction has been made for the sub-domain of ubiquitous computing rooms.

1.4 Overview of the Dissertation

This dissertation is broken up into the ten chapters, with this text falling in Chapter 1, the Introduction. The chapters are as follows:

- **Chapter 1 – Introduction:** The introduction to the dissertation.
- **Chapter 2 – Background Information and Motivation:** Material that gives a context and necessary background to understand the remainder of the dissertation, and material that helps to motivate the rest of the dissertation.
- **Chapter 3 – The Nature of an Interactive Workspace:** Presents the characteristics of interactive workspaces that are most pertinent to coordination infrastructures. This represents contribution **C1** of this dissertation.
- **Chapter 4 – The Event Heap Model:** Describes the properties necessary for a coordination system in an interactive workspace, and how the features of the Event Heap,

an extended tuplespace model, satisfy those properties. This represents contribution **C2** of this dissertation.

- **Chapter 5 – The Event Heap Prototype Implementation:** Describes the prototype implementation of the Event Heap model. Specifically, its design and API are described. This represents contribution **C3** of this dissertation.
- **Chapter 6 – Evaluation:** Evaluates the three contributions of the dissertation in the following ways: a description of the usage of the Event Heap in specific developed applications and deployed environments, a survey of developers and administrators that have used the Event Heap prototype implementation, and a presentation of performance numbers for the prototype implementation.
- **Chapter 7 – Discussion:** Discusses observations about the Event Heap model and prototype.
- **Chapter 8 – Related Work:** Presents work by other research groups that is related to the Event Heap in some way, and describes similarities and differences between that work and the results presented in this dissertation.
- **Chapter 9 – Open Questions and Future Work:** Describes some of the issues that are not yet resolved, and possible directions for continued research related to coordination systems for ubiquitous computing rooms.
- **Chapter 10 – Conclusion:** The conclusion to the dissertation.

Certain conventions are followed throughout this dissertation. First, much of this work has been presented in various conferences or published as journal articles. Where this is true, a note is made in the text. Second, in part to simplify reuse of text from these publications, ‘we’ is used instead of ‘I’ in even when it describes work done solely or in large part by the author. In all cases where assistance was rendered from others, a notation is made in the text.

Chapter 2 – Background Information and Motivation

This chapter presents background material that is helpful for understanding and helps motivate the rest of the dissertation. The material is presented in four different areas. First, the results of studies by various researchers on how humans function in various environments are presented. These studies help inform the characterization of interactive workspaces. The next section presents background material on computer supported cooperative work (CSCW). The collaborations that we hope to support in interactive workspaces fall into the domain of CSCW. Next, to more clearly specify the role of a coordination infrastructure in an interactive workspace, we provide a more detailed look at the nature of "coordination." Finally, we give an overview of the four major coordination system categories that we considered as a basis for our coordination model.

2.1 Ethnographic Studies of People and Environments

This section describes the work of three different research groups on how people interact with their environment. The first sub-section describes work on evaluating how teams collaborate in team-project rooms, both traditional and ones that have been minimally augmented with technology. The second sub-section describes a study of how people incorporate technology into their lives, specifically in the context of homes, and the final sub-section discusses a study of "extreme collaboration," a technique where a group of experts in some field come together in a technology augmented environment to rapidly solve problems.

2.1.1 Team Project Rooms

In [50], Covi presents a survey of dedicated project rooms in use at nine different corporations. A dedicated project room is one that is permanently in use by some working team, and therefore

need not be cleaned up to allow others to use the space. This allows artifacts, like large organization charts, to be permanently left in the room and used on an on-going basis as a focus of the project being done. Aside from the ability to have these long-term artifacts, other benefits of dedicated project rooms include: increased learning by example, better motivation, and improved team coordination. Note that although some of the rooms had electronics or computers in the environment, the focus of this study was on the project teams and not the technology in use, or how to improve that technology.

In particular, certain artifacts and information were found to be helpful for ongoing display in the team-project rooms. Work in progress displays helped teams keep track of their status, and task lists along with task status helped them to identify critical activities. Reference material and past accomplishments were also important public pieces of information displayed in these environments.

They also found that large projected displays could be useful. They specifically said the following about one particular company that used a war room with projected display:

> "The most popular display feature of this room was the computer projection system which they used to develop ideas via a scribe at the in-room computer. Team members felt that a key feature of this technology was the ability to argue at the projected display rather than at each other. By focusing discussion on the representation of the work rather than at a disagreeing team member, they felt that they could better able [sic] to resolve differences and move on with the work."

This work helps provide motivation for investigating interactive workspaces, and in turn the infrastructure for applications that will run in them. As just mentioned, they provide evidence that environments rich in display space can be helpful. One drawback of dedicated project rooms is that the space cannot be shared with different groups since it is difficult or impossible to switch the permanent artifacts in and out of an environment. Dedicating the space is an expensive proposition, so corporations can only afford to provide such environments to teams doing critical work. Interactive workspaces hold the promise of allowing multiple groups to share one dedicated space by saving the state of the shared information displays and restoring them for each team that uses the space. This could allow the benefit of a dedicated room to be shared by more project groups within corporations.

2.1.2 Technology in People's Lives

In [79], Hughes presents an ethnographic study of technology usage in homes in the United Kingdom. While we are not focusing on technology in home environments, their results are

potentially more general to how people interact with technology. The study draws the conclusion that technology must not interfere with daily routine, but instead be adaptable to that routine. At the same time, it must also be easy to reconfigure the technology to allow different uses of a space. Finally, the technology should be easily adaptable to the unique usage styles of different individuals.

One specific result was that a non-trivial element of becoming comfortable and turning a "house" into a "home" was allowing it to be customized to the dwellers tastes. This suggests, and is supported by the previously mentioned study on team project rooms, that the ability to customize an interactive workspace for each group that uses it could be helpful in making people more productive in the space.

Another observation made in the paper is that technology only becomes successful to the extent it is not obtrusive. They comment on television, for example:

> "It is to be noted then, as a corollary of such notions, that the success of the television and its ubiquity in the nations [sic] [United Kingdom's] households can be in no small measure due to its ability to support a wide range of viewing regimes, imposing little or no obligations on the household to mould to its requirements, but rather lending itself to and becoming resolutely intertwined with the routine scenarios of everyday existence, whatever they may be."

They noted that by comparison computers were not a part people's lives and were put in separate rooms or nooks, and engaged only for a specific task. This observation fits with Weiser's [142] definition of ubiquitous computing being the age where computers disappear into the background and become part of people's everyday lives. A good goal for interactive workspaces is to allow teams to work without considering themselves to be engaging the "room computer," but instead just working on a problem and using the computational resources in a natural and intuitive manner. Providing for this fluid interaction with sets of applications while still allowing coordination is the goal of the infrastructure we are presenting in this dissertation.

2.1.3 Extreme Collaboration

In [106] and [105], Mark describes a study she did on *extreme collaboration*. She describes extreme collaboration as: "Working within an electronic and social environment that maximizes communication and information flow." The study looked at Team X, a 16-person NASA team at the Jet Propulsion Lab (JPL) that designs space missions during sessions in a technologically rich war room environment. The team is able to be very efficient in this environment due to a combination of human networking, electronic networking, the ability of individuals to maintain peripheral awareness of other activity in the room, and the ability to quickly form and dissolve

sub-groups to solve smaller problems. The amount of time required to create a space mission design was reduced from 3-6 months to about 9 hours over the course of a week by switching to the war room approach. Previously the team had weekly meetings followed by recesses where team members would work on their own calculations.

Mark distinguishes war room environments from control rooms. A war room is a team collaboration room where all work takes place in the room—team members never recess to another location to come up with intermediate results. This allows team members to continually move back and forth between individual work, small group work, and orchestrated entire teamwork. In a control room, in contrast, the workers are responding in a defined fashion to some continuous stream of external events (e.g. air-traffic controllers getting a continuous stream of new planes to track). Therefore, in a control room the team is event driven while in a war room there is some desired finished product resulting from the work. This distinction is similar to the one that Panko [116] makes about office types, where some do routine work that can be automated, and some do work which is always different and can only be mechanized.

Mark gives five reasons for why collocation was important in allowing the team to be so efficient:

1. Information on changing requirements is immediately available.
2. Individuals have easy access to one another.
3. More members track changes so errors can be quickly caught.
4. Collocation allows the continual communication necessary to discuss alternatives and question assumptions.
5. Communication needs to keep pace with the speed of the design activity.

Point two was particularly important in that not only was there the opportunity for explicit interactions with other people, but also a peripheral awareness of what other people in the room were doing. Specifically she says, “What triggers Team X members to group together is that they continually monitor information in the room both visually and aurally” (this ability to pick out the conversation of interest from many going on in a room is known as the cocktail party phenomenon [42]). Once they hear or see something related to their work, they begin a conversation or a sidebar with the other appropriate parties.

The papers also give a more detailed breakdown of the six important features of *extreme collaboration*:

1. Permanent team of experts.

2. Human and electronic networking are fundamental components of the process.
3. Team interdependencies made visible by the war room.
4. Electronic artifacts are dynamic (e.g. screen displays can change).
5. Physical space allows team member to see one another.
6. Time pressure to produce results.

Points two and four are what prevent extreme collaboration from being possible in a non-technologically enhanced team project room.

The setup for Team X, however, is not general. The technology and room they use were specifically set up for solving the task of designing space missions. Among other technologies, they use several public displays, database programs that allow access to data on past space missions, satellite orbit visualization programs, and a custom publish-subscribe system that allows individuals to push and pull spreadsheet data to and from other workstations in the war room.

A goal for the Interactive Workspaces project is to allow the kind of time-savings that Team X was able to obtain for space mission design to be realized by other project groups. More specifically, we would like to enable this without requiring groups to custom design the hardware and software systems they need to use for the problems they are addressing. In part, this means providing a more general mechanism to allow application coordination.

2.2 Groupware and Computer Supported Cooperative Work (CSCW)

Any collaborative or cooperative activity in which humans engage with the assistance of computers is referred to as Computer Supported Cooperative Work, or CSCW. Supporting collaboration in technologically enhanced spaces is the goal of the Interactive Workspaces project, so at least the human-computer interaction (HCI) aspects of the research fall into the domain of CSCW. It follows that systems infrastructure to support the HCI work needs to take into account the nature and characteristics of this domain.

Groupware is the term within the field that has come to describe specific programs and software systems that are designed to support CSCW. In [57], groupware is more specifically defined as: "Computer-based systems that support groups of people engaged in a common task (or goal) and that provide an interface to a shared environment." The shared environment may either be virtual, or an actual space and people need not be engaged in the task at the same time. Johansen [83] and others have proposed dividing CSCW, and by extension groupware, into four different

types based on location of participants and simultaneity of interaction. These are: same-time, same-place work; same-time, different-place work; different-time, same-place work; and different-time, different-place work. The following table summarizes this and gives examples of each category:

	Same-Place	Different-Place
Same-Time	Electronic Whiteboards	Video Conferencing
Different-Time	Group Bulletin Board	E-mail

Table 1 – Different Types of CSCW and Groupware

The Interactive Workspace research falls solidly into the same-time, same-place category of CSCW and groupware.

In his article [57], Ellis identifies some of the fields that must contribute to the design of Groupware applications and systems. They are HCI, artificial intelligence, social theory, communication and distributed systems. The Interactive Workspaces group has been looking at all of these areas with the exception of artificial intelligence. This dissertation specifically addresses the communication and distributed systems aspects of groupware.

Readers interested in acquiring a broader perspective on work in the field of CSCW over the years are encouraged to look at [71].

2.3 Coordination

Since the goal of this dissertation is to provide a "coordination infrastructure for interactive workspaces," it is worth spending some time more carefully defining coordination, and looking at what other researchers have found about coordination.

To start, what is the definition of "coordination"? Coordination is a very general term, and Malone [104] lists a set of academic disciplines that range from Computer Science to Sociology to Linguistics that deal with some form of coordination. According to the American Heritage Dictionary [1], coordination is "the act of working together harmoniously." Various researchers have tried to come up with other definitions, in many cases to make them closer to how the term is used for HCI or computer work. Here is a list of a few definitions of coordination, with a focus on definitions related to Computer Science (many of these were compiled by Malone in [104]):

- "The additional information processing performed when multiple, connected actors pursue goals that a single actor pursuing the same goals would not perform." [103]
- "*Coordination* refers to the mechanisms by which a community of agents is able to work together productively on some task." [107]

- "The act of managing interdependencies between activities performed to achieve a goal." [104]
- "The operation of complex systems made up of components." [114]
- "Information processing within a system of communicating entities with distinct information states." [114]
- "The joint efforts of independent communicating actors towards mutually defined goals." [114]
- "Composing purposeful actions into larger purposeful wholes." Holt, A. in personal communication to Malone, T.W. for [104]
- "Activities required to maintain consistency within a work product or to manage dependencies within the workflow." [51]

Overall, it can be stated that coordination has to do with communication for facilitating collaboration of a group of agents, either humans or computer applications, as they work together on a task. As such, the goal of a coordination infrastructure is to provide a set of mechanisms that facilitate interactions and minimize coordination overhead.

Of course, the topic of this dissertation is more closely focused on coordinating the interactions of a group of computer programs. In this respect, the work is most closely related to what Gelernter [65] calls a *coordination language*. He specifically says that "a coordination language must allow one active agent to convey information to another whose state is evolving and unpredictable." In his case, a coordination language is a set of primitives in a language that allow specification of coordination, and, when compiled into a set of applications, allows those applications to interact [67]. Papadopoulos [117] more generally defines a *coordination model* as "the glue that binds separate activities into an ensemble." Our goal for interactive workspaces is not to provide language primitives, but rather an API and libraries for applications that enable coordination. In this sense, we are looking at coordination models rather than coordination languages.

Many different models of coordination have been proposed and used in Computer Science over the years. For this dissertation, we consider the four major categories that are commonly used in ubiquitous computing: tuplespaces, RMI/RPC with rendezvous, publish-subscribe, and Message Oriented Middleware (MOM). Low level protocols such as pure sockets or distributed shared memory impose a large amount of coding overhead on developers and are not considered, nor are

coordination techniques intended primarily for high performance computing, such as message passing.

Here is a brief description of each of the coordination models that will be considered:

- **Tuplespaces:** Applications coordinate with one another by exchanging messages, called tuples, through a long-lived central memory known as a *tuplespace*. Each tuple is a collection of type-value fields. Applications may place tuples into the tuplespace, or read a copy of a tuple by specifying a template. Read operations may take copies of tuples or be destructive.
- **RMI/RPC with Rendezvous:** Applications communicate using procedure calls or method invocations that appear to be local but are actually applied to a remote application. The system handles packaging of the parameters being sent to the remote machine, as well as returned values. A rendezvous service, which returns handles to procedures and objects, is provided for applications to locate remote services with which to interact.
- **Publish-Subscribe:** Applications send out messages with different topics to the set of other applications with whom they are coordinating. Consuming applications select the types of messages they wish to receive and are delivered copies of the appropriate messages, regardless of which application generated them.
- **Message-Oriented-Middleware (MOM):** Typically similar to publish-subscribe, except messages are delivered over Internet scales and are typically routed through several intermediate servers along the way to their destinations. MOM systems usually provide for transactions, guaranteed delivery and other semantics important for business-to-business inter-application coordination.

All of these coordination systems are described in more detail in Chapter 4, where we describe the important properties for coordination systems in interactive workspaces and explain which properties each coordination model possesses.

Chapter 3 - The Nature of an Interactive Workspace

Few interactive workspaces exist, and those that do are mostly research prototypes. Therefore, to reason about the type of coordination infrastructure needed in such a space, we collaborated with researchers in other fields and discussed with them how they would foresee themselves using such a space. In addition, we built the previously mentioned iRoom, and ran our own research meetings in the room, developing software to make things run more smoothly. We have also read studies [50, 83, 105, 106] on the use of low-tech team-project rooms where groups come together to solve problems using white boards, flip charts, and other non-electronic equipment (these are described in more detail in Chapter 2).

Summary of Interactive Workspace Characteristics
Based on Human Factors
H1. Bounded Environment
H2. Human Centered Interaction and Flexible Reconfiguration
H3. Human Level Performance Needs
Based on Technology Factors
Heterogeneity
T1. Hardware
T2. Software and Software Platforms
Changing Environment
T3. Short Timescale Change
T4. Long Timescale Change (Space evolution)

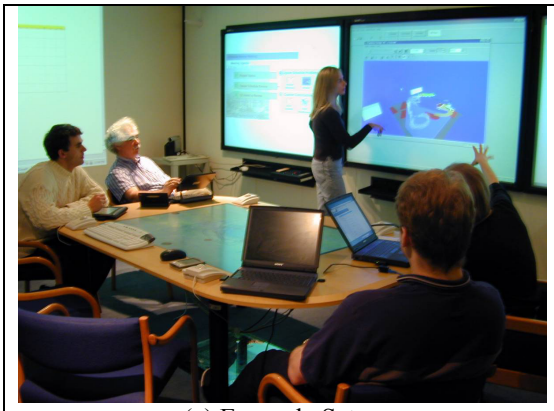
Table 2- Summary of Interactive Workspace Characteristics

These activities led us to two categories for characteristics of interactive workspaces: those based on the human factors of collaboration in a team-project room, and those based on the technology that is likely to be deployed in such spaces. Note that the two are interrelated; for example, interactive workspaces need to provide for user's portable devices entering, engaging and disengaging with ongoing activity in the space. This dynamism arises from both the availability

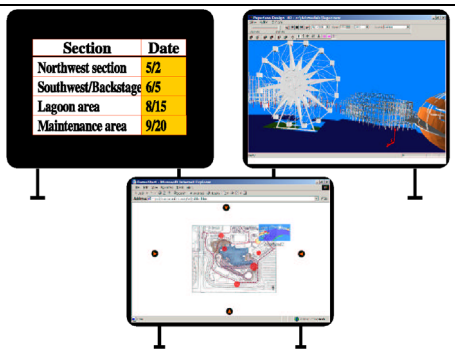
of portable device technology and the way humans use such devices. The chapter begins with a scenario to provide motivation for the human and technology characteristics of interactive workspaces that we present in the following two sections of the chapter. Table 2 provides a summary of these characteristics. The final chapter section addresses some of the common usage modalities we found for interactive workspaces. This chapter is an expanded version of Section 3 of [85].

3.1 Motivating Scenario

Although the rest of this chapter will argue in terms of abstract properties, we give here a scenario that is set in the iRoom, which reflects how such spaces may used and that helps to motivate our arguments. The scenario is based on research we have done with the Center for Integrated Facilities Engineering (CIFE) [102], a civil engineering research group on campus. A more detailed video version of this scenario may be found on the supplementary CD-ROM included with the dissertation., or on-line at: http://iwork.stanford.edu.



(a) Example Setup



(b) Abstract Representation of Three of the Applications Displaying on Different Screens

Figure 4 - Construction Management in the iRoom

Consider a group of construction management engineers and contractors using the iRoom to plan a major construction project. 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.

As the meeting begins, the leader displays a web-page outline that will serve as a guide for the meeting on the left-most touch screen. Each topic in the outline is a hyperlink that brings up related data for that topic on the other displays in the room. Some of that data is in the form of web pages, while other data is brought up in specific construction site modeling and planning applications, some of which were not originally designed to run in the workspace. Figure 4

shows a mockup of the iRoom being used for such a scenario and an abstract representation of what might be on the touch displays in the iRoom.

Later in the meeting, the leader is suddenly called away and turns off his laptop that is being displayed to one of the large touch screens in the room. No other software that had been interacting with files and applications on the leader's laptop is affected, but the display screen he was using is now blank. Another member of the group uses his wireless PDA to switch it to display a machine that is permanently in the workspace.

The leader leaves the group with the task of figuring out a way around a problem with the 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 (and is therefore called a "4D-Viewer"), and the project scheduling software on another. All of these are separate stand-alone applications, but each application's data is automatically associated with similar data being displayed in the other applications. Thus, when the users select or make changes in one view, the other views immediately reflect the new state, and similarly the model viewer and schedule keep their dates synchronized. They decide to open a second 4D-Viewer on a third screen in the room and dissociate its date from the other applications. They set it to a date earlier in construction so that they can do side-by-side comparisons with the construction state for other dates. Still, as they select different parts of the model to look at, both the viewer with fixed date and the original synchronize to the same view.

When the meeting is over, the users shut down the room using a simple web based interface.

3.2 The Human Side

Human side characteristics arise from the way humans interact in and perceive team-project rooms.

3.2.1 Bounded Environment

An interactive workspace is bounded in physical extent, therefore humans expect devices and applications to coordinate with one another within the space. Similarly, coordination with applications and devices outside of the space should not occur unless a user specifically requests the coordination. In the scenario from the beginning of the chapter, for example, the construction engineers expect their laptops to interact with the room infrastructure while in the workspace, but not when they leave (when the leader turns on his laptop back in his office, his interactions with it should no longer effect the software running in the workspace).

Ellis found that for CSCW systems, the "user interfaces should smoothly mesh the access model with the user's conceptual model of the system" [57]. In the case of an interactive workspace, this means that device and applications by default should only interact with other devices and applications in the local environment. Armando Fox, a principal investigator of the Interactive Workspaces project, has previously identified this as the 'boundary principle' in a paper with Tim Kindberg [95]. Therefore, the software infrastructure for a particular room must only support the devices within the room unless explicitly over-ridden by users to do otherwise. We refer to the bounded nature of an interactive workspace as **H1**.

From a systems perspective, one challenge of addressing **H1** is that traditional networks and administrative domains do not typically conform to architecturally bounded regions such as rooms. Systems that use broadcast on a sub-net to coordinate applications can fail to behave in a fashion users might expect if the sub-net spans several rooms, a floor, or even a whole building. Wireless networking solutions can also cause problems since there is no simple way to constrain the propagation of the radio waves from a base station to a single room. Any infrastructure for interactive workspaces needs to overcome this scoping problem, even if it is by simply asserting that devices should not be connected to the coordination framework if they are not within the corresponding room or workspace (this method is used in the Event Heap model). This need to scope based on real world location was one of the motivating factors for the design of the Intentional Naming System [19]. In their case, they wanted to provide a method for users to name target devices based on their characteristics, including real-world location. The problem of automatically and reliably determining real world location and associating and disassociating devices from the infrastructure in the location is separable from construction of the coordination infrastructure, and a challenging research topic in its own right. The coordination infrastructure presented in this dissertation does not attempt to address this issue.

3.2.2 Human-centered Interaction and Flexible Reconfiguration

Work in team-project rooms is driven by a group of people working through one or more problems. As work proceeds, different tools and information may be necessary, and different sub-groups may form [105, 106]. In standard team-project rooms this means that the tools at hand are constantly being used in different ways—white boards are erased and written over, flip charts set to different graphs, etc. By extension, we expect that participants in an interactive workspace will use the devices and applications that best help them to solve each new problem that arises in the course of a collaboration session. As Streitz puts it in [136]: "The room should have the character of a market place or a landscape providing opportunities for spontaneous

encounters and informal communication." In summary, the coordination system needs to support flexible reconfiguration and dynamic coordination of components being used in the space.

We refer to the desire to keep interaction centered on humans and the ability to flexibly reconfigure applications and hardware in a workspace as characteristic **H2**.

Social Processes

Our intent in providing our infrastructure and applications from the Interactive Workspaces project as a whole, is to provide for collaboration among users of a space that is mediated primarily by social processes, rather than a process imposed by the software in the room. This is in part motivated by the findings of Ellis in [57] that leaving users' interactions with the software and each other to social processes can be helpful in a CSCW setting:

> "Leaving the processes to social protocols encourages collaboration: the group must develop its own protocols, and consequently the groupware itself is more adaptive. Social protocols (in particular ad hoc protocols), however, can be unfair, distracting, or inefficient. In contrast, embedding a group process in software as a technological protocol ensures that the process is followed, provides more structure to the groups activity, and assists less experienced users. Technological protocols can be overly restrictive: a group's idiosyncratic working style may not be supported, and the system can constrain a group that needs to use different processes for different activities."

He also noted that the technology must provide some constraints, but that a fine line needed to be walked between social processes and structured technology. Roseman [128] and Harper [77] have also found reliance on social processes to be useful in their work on CSCW.

Flexible Sub-group Formation

One key aspect of human centered interactions in these spaces is the need to support dynamic formation of sub-groups in the space, as well as the subsequent remerging of sub-groups back into the whole. Any coordination infrastructure needs to provide enough flexibility to support the rapid changes that are entailed by such changes in group interaction. As Streitz puts it in [138]:

> "In a time-critical situation, it would be very useful if one can reduce [the] cycle time of full team meetings [followed by] subgroup meetings. An alternative is to provide ways for subgroups to split up *during the meeting* in the same room, do their work, rejoin and then immediately merge the results. **Providing adequate information technology support for this scenario requires a team or project room which is equipped with components and resources which are very flexible so that they can be reconfigured dynamically and on-demand in order to meet the requirements of changing team work situations.**" [Bold emphasis added]

Other researchers have also found that sub-group formation is critical to successful collaborative work of teams working in team project rooms, including [83], [136], and [50].

Application Ensembles

Gelernter proposes *ensembles* in[65]. He characterizes them in the following way:

> "Ensembles are the best adapted inhabitants of the evolving hardware environment: the densely intertwined computer jungle that is taking root everywhere. In this developing environment, computations will rarely stay cooped up inside a single computer. They will interact with—draw services from and supply services to—other computations on other machines."

From the systems perspective, the flexible nature of human centered interaction in interactive workspaces is that the software environment will consist of ensembles of applications that dynamically coordinate with one another. This style of interaction is evident in the scenario we gave at the beginning of the chapter where the construction engineers pull up various applications to better understand various problems.

3.2.3 Human Performance Constraints

One key constraint for any coordination infrastructure in an interactive workspace is that it should not introduce delays that would impact the smooth flow of users' work. In his paper on Groupware [57], Ellis puts it this way: "Real-time systems supporting [synchronous] activities must not hinder the group's cadence." Terry Winograd, a principal investigator on the interactive workspaces project, breaks down the specific areas that must be made sufficiently fast for users into the following four categories [144]:

1. Input observers (device drivers) can sample fast enough for timing conditions (e.g., the sampling rate of a positioning device is quick enough to appear perceptually smooth).
2. Output devices can update fast enough for timing conditions (e.g., guaranteed frame rate for visual rendering).
3. Data can be transmitted between input and output devices fast enough for timing conditions (e.g., may need to send a new set of coordinates, versus sending an entire image for each change).
4. Computation per observation is fast enough for timing conditions (e.g., pointer draw must take less time than the period between subsequent samples of the mouse position).

Since workspaces will be used by humans, there is also a limitation on the number of participants that can meaningfully collaborate within a given room. Most studies have shown that meaningful

collaboration is limited to groups of two to fifteen participants [83]. This observation can be used to establish a limit on the number of devices and amount of coordination traffic an infrastructure needs to be able to handle.

Additionally, humans have certain expectations about reaction to events generated through their interaction with the system. For example, an event generated to turn on all lights is only relevant for a few seconds, after which lights that come back on-line shouldn't turn themselves on.

We refer to the need for system performance to be tailored to human needs as characteristic **H3**.

3.3 The Technology Side

3.3.1 Heterogeneity

One of the main characteristics of an interactive workspace will be the heterogeneity of both devices and software in the space. While it is possible to custom build an interactive workspace and a suite of applications using a standard set of interoperable devices running the same development platform (for example, Java across a set of standard PCs and Windows CE machines), this precludes evolution of the space to allow integration of new devices, and excludes the possibility of integrating existing applications not built on the platform of choice. As a general principle, the GaiaOS project, another research project investigating ubiquitous computing rooms, has also found heterogeneity to be an important characteristic to address [39].

Devices

The devices in an interactive workspace will naturally be heterogeneous because each will be chosen for its suitability to addressing a particular kind of task. In the scenario presented at the beginning of the chapter, for example, individual engineers had laptops and PDAs they brought with them, but they also used permanent machines in the space with large touch screens. In other cases, there may be some more task specialized devices that need to be used. Perhaps a certain wireless device is well suited to attacking certain problems for the collaborators using the space. Some collaborative activities may require the use of an exotic or custom-built device like a high-resolution custom display such as the interactive mural [74, 81]. This type of hardware heterogeneity we refer to as characteristic **T1**.

Other researchers have also found supporting heterogeneous collections of devices to be a key aspect of ubiquitous computing. In their paper on the Microsoft Easy Living project [34], Brumitt et al. posit that "to support disaggregated computing, many of the traditional activities of an operating system must be supported across a distributed heterogeneous set of networked

devices." In an early paper on parallel computing in general [29], Bisiani offers the argument that multiple machine types need to be supported since some languages run better on certain platforms. He also offers that specialized devices, such as array processors, may be needed to assist with certain types of problems. [29].

Both Esler [60] and Rekimoto [124] believe that future devices will be chosen for function and convenience—there will be no more Swiss Army Knife, do everything computers. These simpler devices will then be combined together to achieve some task, much as we use several physical tools in conjunction to accomplish real-world tasks. Rekimoto further argues in [125] that even if heterogeneous support is restricted to Macs, PCs and Unix, each have their own advantages and disadvantages, and users will need to use all of them to get the maximum utility.

In his original article on ubiquitous computing [142], Weiser sums up the need for heterogeneous devices as he argues for how computers from the size of post-it notes to wall sized displays need to be used together:

> "The real power of the concept [of ubiquitous computing] comes not from any one of these devices—it emerges from the interaction of all of them. The hundreds of processors and displays are not a 'user interface' like a mouse and windows, just a pleasant and effective 'place' to get things done."

Software

Similar arguments apply to the heterogeneity of software components in an interactive workspace. To start with, just supporting the variety of hardware in an interactive workspace will require supporting the various software platforms used by those devices.

Another important reason to support integration of diverse software platforms is to provide support for legacy applications within an interactive workspace. Working groups within companies rely on complex off-the-shelf software as well as in-house solutions (the construction engineers in our scenario from the start of the chapter, for example, rely on a custom developed "4D-Viewer" that displays a 3D model of construction state for any specific date during construction), and it is usually impractical or impossible to rewrite the applications just to make them work in an interactive workspace.

The need for legacy support has been found to be important by other CSCW and ubiquitous computing researchers. In Falkowski's Room Management System [62], for example, they found that "one important aspect was the user need of using standard software, e.g., Office-Suites on MS-Windows platform." Roseman reports [128] that one of the features they wished they had had in their GroupKit groupware system was support for conventional legacy applications. Both

the Information Bus [115] publish-subscribe system and the Context Toolkit [54] included legacy support as design goals.

To provide users with flexibility in the software tools they use to solve problems, it is also important to support integration of products from various vendors, which may all have different development platforms. The coordination infrastructure needs to support these platforms to allow users to compose the applications. This argument has also been made by [60].

A final reason to support different software platforms is that some languages may be better suited for coding of different application types. Bisiani proposed in [29] that a good technique might be to use a poor-performing, rapid-prototyping language to start, and then replace key portions with components in more efficient languages. Dey proposes that, in regards to his work with the Context Toolkit [54], embedded devices may all have their own SDKs that need to be supported. While the intent of the Event Heap model is not to specifically support context, this argument nevertheless holds true for embedded devices that may be coordinated by users in an interactive workspace.

We refer to the variety of software environments that will be present in an interactive workspace and the need to support them as characteristic **T2**.

3.3.2 Changing Environment

Another important characteristic of an interactive workspace is the constant change within the space. This change will occur both on short time scales as devices crash and restart or as devices enter and exit the workspace, as well as on longer time scales as the interactive workspace as a whole is continually upgraded and modified.

Short Term Change

Short-term change occurs in two main ways. The first is through the entry and exit of portable devices that are brought in by users of the space—the meeting leader abruptly leaving in the scenario from the beginning of the chapter, for example. The applications and capabilities of these devices need to be integrated with other software in the space as smoothly as possible, and their unexpected exit should have no ill effects on the other devices and applications in the workspace (aside from capabilities of that device no longer being accessible). Even in normal operation, "experimental" devices or software often fail unexpectedly and must be recovered; the coordination infrastructure needs to be tolerant of these failures as well. The characteristic of changing over short time scales we refer to as **T3**.

The short time scale dynamism of ubiquitous computing has been noted by other researchers as well. In his original article on ubiquitous computing [142], Weiser noted that: "Pads [PDA sized devices], tabs [tablet computers] and even boards [white board sized devices] may come and go at any time in any room, and it will certainly be impossible to shut down all the computers in a room to install new software in any one of them." Esler [60], Dey [54], and Cerqueira [39] have also noted that it is important for ubiquitous computing infrastructures to be tolerant of devices crashing, entering and leaving spaces.

Long Term Change and Space Evolution

Long-term change occurs in the evolving layout of the interactive workspace and the complement of devices permanently embedded in the space: as the space is used to solve new problems, obsolete devices are removed, and new technology is brought into the space. This is similar to the 'incremental evolution' that [56] suggests will take place in smart homes. Any coordination infrastructure must therefore be capable of being adapted to work with new devices and platforms over time, thereby allowing coordination between old and new applications and devices.

The importance of taking evolution into account in the design of computer systems in general has been gaining support lately. For the Recovery Oriented Computing initiative [118], Patterson et al., propose that the monitor ACME, or Availability, Change, Maintainability and *Evolutionary growth*, should be used to evaluate utility of future computer systems. Researchers on other specific systems such as the Information Bus [115], and the infrastructure being developed by the Portolano Project at the University of Washington [60] have also listed support for evolution as being key.

From the perspective of CSCW work, Johnson-Lenz [89] has indicated that evolution is not just something expected to occur over time, but actually a part of the long-term problem solving process: "For ongoing group work, the design process must be dynamic and evolve with the group's needs and activities." Arias [23], in a paper on the EDC (Envisionment and Discovery Collaboratory), gave the following principles for how evolution factors into CSCW work:

- Systems must evolve; they cannot be completely designed prior to use.
- Systems must evolve at the hands of the users since they will know best what needs to change.
- Systems must be designed for evolution.

- Evolution of systems must take place in a distributed manner, across different devices, platforms, etc.

We refer to the incremental evolution that occurs in interactive workspaces as characteristic **T4**.

3.4 Important Usage Capabilities

Besides the basic characteristics of interactive workspaces, it is also important to understand what activities users are likely to carry out in an interactive workspace. Through our own use, and through consulting with collaborating research groups (for example, CIFE, a civil engineering group, the Center for Integrated Facility Engineering [101]) we came up with three major user modalities: moving data, moving control, and dynamic application coordination. At least two of these were also thought to be important by Myers [113], who states with regards to the Pebbles multi-device integration project: "The challenge for the design of applications is to show only the appropriate information in each place and *allow the fluid transfer of control and information* among private and shared displays" [emphasis added].

3.4.1 Moving Data

Users in the room need the ability to move data among the various visualization applications that run on screens in the room and laptops or PDAs that are brought into the workspace. This includes provision for translating data formats to allow display on the various devices.

Other researchers have found data movement facilities to be important in multi-device situations. [125], [124], and [136] stress the need to provide a convenient and intuitive means of moving data between devices in multi-device situations, and propose several mechanisms to do so. Regarding the importance of data movement, Rekimoto states in [124] that only "once data transfer between multiple devices becomes widely available, [will we] have greater flexibility in designing a physical work environment by combining multiple digital devices."

One important aspect of data movement that arises in interactive workspaces is the need to provide for both public and private space, as well as a means to transfer information between the two. By allowing these two types of space, users can work privately on something until they feel it is suitable for group viewing, and then transfer it off their personal device onto a communal display in the environment. A survey presented by Streitz in [137] found this to be a common technique. Other researchers including Greenberg [70] have investigated how people use public versus private space and computer mechanisms to support this.

3.4.2 Moving Control

In an interactive workspace users no longer sit in front of a single display, so controlling applications and devices becomes more difficult. While it is possible to force people to walk over to each device to control it, this introduces interruptions during meetings. Instead, a means should be provided to allow any device in the workspace to be controlled from any other device. In particular, for standard GUI interfaces there needs to be a means to provide mouse and keyboard control.

Other researchers have also identified providing remote control of applications as being important in ubiquitous computing environments. Esler states that they are trying to provide people multiple interfaces to services from different devices [60]. In his paper on the i-Land system [136], Streitz identifies as a key issue how users will be able to interact with all of the devices in a space. In an earlier study [137], Streitz also found that users wanted methods to control the main displays in a ubiquitous computing room so that they could modify the contents being shown.

3.4.3 Dynamic Application Coordination

The specific applications that teams need to display data and analyze scenarios during problem solving sessions are diverse,[1] and any number of these programs may be needed during a single meeting. The activities of each tool should therefore coordinate with others as appropriate. For example, the financial impacts of a design change in a CAD program should automatically show up in a spreadsheet program showing related information running elsewhere in the room. We call this style of interaction dynamic application coordination.

[1] Bechtel Corporation, for example, reports that they use over 240 software tools during planning, construction and follow up for construction projects [93].

Chapter 4 – The Event Heap Model

This chapter presents the Event Heap, a system infrastructure model that provides for inter-application coordination given the characteristics of interactive workspaces presented in the previous chapter. This chapter is divided into two major sections and one smaller comparative section. The first section describes a set of properties that are crucial for any coordination infrastructure for an interactive workspace. The second section describes the features of the extended tuplespace model called the Event Heap that cause it to possess the aforementioned properties. A final section looks at some alternative coordination models and shows why they were not suitable starting points for a coordination model in an interactive workspace.

This chapter is an extended version of Sections 4, 5, and 6 of [85].

4.1 Properties for Coordination Infrastructures

The characteristics of interactive workspaces discussed in Chapter 3 imply a set of properties that a coordination infrastructure for such spaces must support. Table 3 provides a summary of these properties and how they relate to the characteristics discussed in Chapter 3, while the rest of the section presents this same material in more detail.

System Property		Interactive Workspace Characteristic Supported
P1.	Limited Temporal Decoupling	• *Human Level Performance Needs* (**H3**) by allowing data to disappear after their period of relevance is exceeded. • *Short Timescale Change* (**T3**) by masking transient failure, and by preventing system performance degradation by limiting build-up of tuples.
P2.	Referential Decoupling	• *Human Centered Interaction and Flexible Reconfiguration* (**H2**) by minimizing need to hard-wire specific configurations. • *Short Timescale Change* (**T3**) by discouraging application interdependence, thus minimizing the chances of cascading failures.
P3.	Extensibility	• *Human Centered Interaction and Flexible Reconfiguration* (**H2**) by making it easier to integrate diverse applications. • *Long Timescale Change* (**T4**) by allowing applications to be adapted as workspace evolves.
P4.	Expressiveness	• *Human Centered Interaction and Flexible Reconfiguration* (**H2**) by providing for a variety of coordination patterns. • *Heterogeneous Software and Software Platforms* (**T2**) by providing a variety of coordination patterns that may be needed by legacy software. • *Long Timescale Change* (**T4**) by providing for new coordination patterns that may be needed with future applications.
P5.	Simple and Portable Client API	• *Heterogeneous Hardware* (**T1**) and *Heterogeneous Software and Software Platforms* (**T2**) by minimizing effort required to support new hardware and software platforms. • *Long Timescale Change* (**T4**) by minimizing effort required to support new platforms.
P6.	Easy Debugging	• *Human Centered Interaction and Flexible Reconfiguration* (**H2**) by making it easier to debug user problems. • *Long Timescale Change* (**T4**) by making it easier to troubleshoot integration of new technology.
P7.	Perceptual Instantaneity	• *Human Level Performance Needs* (**H3**).
P8.	Scalability to Workspace-sized Traffic Loads	• *Bounded Environment* (**H1**) limits scalability to a single workspace. • *Human Level Performance Needs* (**H3**) also restricts needed scalability to traffic humans can generate.
P9.	Failure Tolerance and Recovery	• *Human Centered Interaction and Flexible Reconfiguration* (**H2**) by minimizing impact of failures on users. • *Short Timescale Change* (**T3**) by preventing crashes from causing systemic failure and allowing for quick recovery.
P10.	Application Portability	• *Human Centered Interaction and Flexible Reconfiguration* (**H2**) by encouraging development of a larger set of applications that can be composed. • *Long Timescale Change* (**T4**) by providing a broader selection of new applications with which to evolve workspace functionality.

Table 3 - Necessary Coordination System Properties

4.1.1 Limited Temporal Decoupling (P1)

Temporal decoupling allows communication between components that are not simultaneously active. This allows newly-started applications or devices just entering an interactive workspace to react to activity that occurred in the seconds or minutes[2] before they became connected. It also permits applications that crash and restart to receive communications sent while they were restarting. Temporal decoupling addresses the short-term change (**T3**) experienced in interactive workspaces.

On the other hand, the system should limit temporal decoupling to a 'relevant' time interval; humans expect an action in one application to trigger side-effects in other applications within a reasonable time, or not at all (**H3**).

In addition, limiting temporal decoupling solves the problem of unconsumed messages, which if buffered forever would over long periods of time result in an accumulation of messages, system slow down and, eventually, a system crash.

4.1.2 Referential Decoupling (P2)

Referential decoupling allows entities to communicate with one another without naming each other specifically. Coordination systems with this property encourage the design of applications that are minimally interdependent with one another, but that can nonetheless react to one another. One way that referential decoupling is possible is by forcing senders and receivers to only interact through an intermediary (this is done, for example, in the Metaglue system [47]). Another is for senders to broadcast messages with attributes, and have receivers select messages for receipt based on attributes rather than on the name of the message source (for example, in the Intentional Naming System [19]).

Referential decoupling makes it harder to create applications that are tightly interdependent since applications are not programmed to interact according to some pre-defined pattern with pre-specified peers. As Carriero, et. al., put it in [37], "processes can be designed to know exactly as much about one another as is appropriate for the programming situation at hand." In other words, applications need not depend on specific knowledge about other applications unless that knowledge is related to the task they are pursuing. This provides for short-term changes in

[2] The exact amount of time will depend on the period of relevance of the information to which the reaction is occurring—a "turn on the lights" message should be ignored after a few seconds while a "current meeting topic" message might be relevant for several minutes.

interactive workspaces (**T3**) since applications are less likely to crash as a result of the disappearance or failure of other applications with which they are being coordinated.

In addition to reducing interdependencies, referential decoupling makes it easier to design applications whose components are location independent. Applications no longer need to send update messages to a specific application on a specific machine, but can instead send the message and have the intermediary route it to the appropriate location, or have the receiving application pick it up independent of location based on the content of the message. This flexibility leads to a coordination infrastructure that is more conducive to the dynamic selection of applications in an ensemble, and is thus better able to support human-centered interaction (**H2**) in an interactive workspace.

Referential decoupling for ubiquitous computing room infrastructure has also been found to be important by the designers of the Gaia OS [39]. One piece of their system is "the *Event Manager*, which is used to distribute information among components, while maintaining loose coupling."

4.1.3 Extensibility (P3)

A coordination system needs to provide extensibility in order to cope with the long-term change and incremental evolution (**T4**) of interactive workspaces. It must be possible to add functionality and adapt applications to one another with no or minimal modifications, and without having access to their source code. This allows integration of task specific legacy applications for use in an interactive workspace environment, supporting **H2**. In other work, Vahdat [140] and Hull [80] (in networking and context aware computing, respectively) have emphasized extensibility to allow for evolution.

Some important techniques that should be supported to provide extensibility in a coordination system are:

- **Snooping:** This allows applications to monitor communications between other applications in the system. This technique allows a new program to be integrated with a pre-existing set of applications by having it read and react to messages sent by that set. For example, a smartclassroom application suite that allows the professor to coordinate several electronic whiteboards could have an application added on to it that displays the whiteboard state on students laptops.
- **Interposability:** This technique allows an intermediary application to pick up messages from a source, translate them to a new format, and then forward them on to a receiver that only

understands the new format. This allows applications that communicate using different message protocols to interact with one another without having to modify either program's code. Interposition as a general technique has been advocated also for traditional operating systems. For example, Jones, et. al. [90] advocate a system allowing interposition between system calls and the OS to improve operating system extensibility.

Snooping and interposition are specific examples of the more general technique of *stream transformation*. This technique allows an application to receive messages of several different types that are being sequentially emitted by one or more applications and use the information in those streams to create one or more new message streams. Some important types of transformations are: summarization (a large number of messages are reduced to a stream of fewer messages), interpolation (additional messages are created based on input messages), and stream merging (messages from one or more input streams are combined to create a new output stream) [27].

4.1.4 Expressiveness (P4)

As defined by Carriero, et. al. [37], expressiveness refers to what can be expressed, how easily, and how concisely. An interactive workspace coordination infrastructure and its API should be sufficient to express as many different types of coordination as possible. This need not mean that a separate API call be provided for all potential interactions, but simply that the set of API primitives provided can be used to express a variety of different types of coordination (different routing patterns, synchronous and asynchronous communication, etc.).

This flexibility is needed to provide for integration of legacy applications (**T2**), which may depend on specific types of coordination or distribution patterns. It also allows applications to be arranged to interact with one another in a variety of ways, which supports more flexible, human-centered interaction (**H2**). Finally, it allows for long-term change and incremental evolution (**T4**) of interactive workspaces by making it more likely that future applications and devices can be supported within the framework.

In another situation where diverse devices and changing environments needed to be handled, Adjie-Winoto, et. al. [19], also found expressiveness to be a key feature for the Intentional Naming System.

4.1.5 Simple and Portable Client API (P5)

The number of API calls supported, and the amount of code necessary to support the infrastructure in the client libraries should be minimized. The main reason for this is to simplify the task of porting the client infrastructure code. This makes it easier to support the heterogeneous devices in the space (**T1**), and to support new platforms that may be integrated with an interactive workspace as it evolves (**T4**). The client libraries also need to be small to insure they will fit on impoverished devices. Simplicity of the API is also important since it minimizes the amount of code that needs to be written to integrate legacy applications (**T2)**.

Note that although this requirement is seemingly in contradiction with **P4**, which advocates an API that allows many different coordination patterns to be expressed, the two are actually compatible. Just as it is possible, for example, to have a RISC processor with a small number of operations that is fully general in its capability to do computation, it is possible to have a small number of coordination primitives that are sufficient to express a variety of coordination types. This will be discussed in more detail in the second half of this chapter in the sub-section on Infrastructure API.

In support of the tuplespace model (also known as the Linda model after the original tuplespace implementation), which has a very simple API, Bjornson says [30]:

> "Simplicity leads to power in another way: the leveraging power of reuse. The same coordination framework can be incorporated in different languages, applied to problems from different domains, and deployed on different hardware."

Bjornson goes on to say that there is a continual struggle in the systems community on complexity, and while we disagree on his assertion that most researchers tend toward complexity, we agree with his basic contention:

> "Linda's simplicity has also worked against it in some contexts; people with complex, fancy or technically demanding requirements often dismiss Linda because they assume that impressively complex problems demand impressively complex solutions. The research community is split on the issue; a significant minority of researchers has [sic] always sought the simplest systems available, but many researchers continue to prefer complexity on principle."

We believe that the potential for code reuse, portability to other platforms, and the reduced learning curve that result from using a simple API make it a key property to support in any interactive workspace coordination infrastructure.

4.1.6 Easy Debugging (P6)

Since the ensembles that arise from human-centered interaction (**H2**) in an interactive workspace will be composed of applications not necessarily designed to work with one another, the coordination system must be designed to make it easy to figure out and debug interactions. Easy debugging also makes it easier to troubleshoot the integration of new devices and applications into the workspace, and thus support workspace evolution (**T4)**.

Debugging in any parallel computing situation is known to be difficult, so simpler system models and debugging support are important. Among other places, this has been pointed out in [37]. The MIT Intelligent Room project [47] found that debugging applications written using their Metaglue system could be quite difficult not only because of the parallel nature of the environment, but also because infrastructure in ubiquitous computing rooms is long lived and the result of a bug may not be experienced for days or even weeks. Unless the system provides support for tracing data between components and identifying the originator, it can also be difficult to isolate the problem.

4.1.7 Perceptual Instantaneity (P7)

Since the coordination infrastructure is intended to support the interaction of applications with one another as driven by the users of the space (**H3**), coordination and actions across applications and devices should be perceptually instantaneous for the humans working in the interactive workspace. Studies show that perceptual instantaneity varies from 30 ms to about 1 s of latency depending on the type of action taken by the human, or activity in which the human is participating [21]. More specifically, Miller [109] identifies the following thresholds for *R*, the time it takes for the system to respond to a user's command:

- *R>100 ms:* the illusion of "instantaneous" response time is lost; user perceives the system as sluggish.
- *R>1 sec:* the user's thought process is interrupted and the delay is perceived as obtrusive.
- *R>10 sec:* the user becomes distracted from the task at hand and will start to work on other tasks while he waits.

This system property means that the coordination system need not perform at the level of systems that are used to coordinate distributed computation. In these systems, inter-process coordination is the main bottleneck and communication throughput and latencies must be minimized. Relative

to these high performance systems, a coordination infrastructure in an interactive workspace can burn cycles and bandwidth to provide for some of the more challenging system properties.

4.1.8 Scalability to Workspace-sized Traffic Load (P8)

As the coordination system will only work within the bounded environment (**H1**) of the interactive workspace, the system need only scale to handle the amount of load that can be generated by humans working therein. This load is limited by the number of devices and applications in use by humans, and the rate at which they cause the applications and devices to generate coordination messages (related to **H3**). The number of humans is limited by social factors, which constrain the total number of participants that can meaningfully work together in a workspace.

We estimate that interactive workspaces of the near future will have on the order of tens to hundreds of devices, which is in line with Weiser's estimate of hundreds of computers per room [142]. We expect that during meetings humans will trigger coordination events on the order of one time per minute through their interaction with applications. On top of this, we expect there to be status updates from devices and applications at a rate of around ten events per device per minute. In aggregate, we therefore expect there to be on the order of tens to hundreds of events per second that the system will need to be able to handle with a latency of less than 100 ms.

4.1.9 Failure Tolerance and Recovery (P9)

In order to be productive, users must not continually be interrupted during collaboration. This means that a failure in one component should not cause other components or the system infrastructure to fail. Unfortunately, individual applications will be failure prone. This is certainly true for the research software we encounter in the iRoom, but even commercial software has bugs which can lead to unexpected failures. Many studies have shown that buggy software is a leading cause of unplanned downtime in large-scale computing infrastructures [18, 43, 69, 112]. Further, although the infrastructure system should be constructed to be robust, it too may fail on occasion. The system therefore needs to assume that failures will be common and provide mechanisms for coping with them. This helps provide for the short-term dynamism (**T3**) of the workspace. The same mechanisms can support human-centered interaction (**H2**) by minimizing the impact of disruptions caused by failures on the collaborators in an interactive workspace.

In general, for interactive workspaces to ever become widely deployed they need to "just work." It is not realistic to expect a full-time system administrator to keep a workspace running. Users will treat the devices in interactive workspaces as appliances that shouldn't fail in unexplainable

ways. At the same time, many of the devices may be commercial-off-the-shelf (COTS) hardware that are failure prone or were not designed to be integrated with heterogeneous equipment. Thus, failure needs to be anticipated as a common case, rather than an exception [95]. All of this means that the software framework must ensure that failures in individual applications and devices are non-catastrophic, and must provide for quick recovery, either automatically or by providing a simple set of recovery steps for the users. This goal of being able to handle failure gracefully is also one that the Portolano Project has advanced for more general ubiquitous computing infrastructure systems [60].

4.1.10 Application Portability (P10)

The coordination infrastructure and applications that are built on top of it should be deployable in any interactive workspace running the infrastructure. This property should be inherent in the entire design—nothing about the coordination infrastructure should specifically associate it with one particular interactive workspace. Further, the general programming style suggested by the infrastructure should be designed to discourage writing applications that are closely associated with a single interactive workspace. While it is a sound general design principle to encourage compartmentalization and reuse, encouraging applications to be created independent of any given interactive workspace also leads to a larger selection of applications for use in any given workspace. This in turn gives more options for long-term change and evolution of individual interactive workspaces (**T4**), and provides a better human-centered experience (**H2**) by giving users more tools from which to choose during collaborations.

The Gaia OS project has also listed application portability as one of their primary goals for their ubiquitous computing room infrastructure [39].

4.1.11 Property Summary

Table 4 shows how the properties just described allow a coordination system to provide for each of the interactive workspace characteristics discussed in Chapter 3. This table provides the inverse mapping of the one given in Table 3.

Interactive Workspace Characteristic	Supporting Properties or Design Decisions
Based on Human Factors	
H1. Bounded Environment	• Having one coordination infrastructure per interactive workspace scopes all interactions to that space. • *Scalability to Workspace-sized Traffic Loads* (**P8**) is limited by this characteristic.
H2. Human Centered Interaction and Flexible Reconfiguration	• *Referential Decoupling* (**P2**) minimizes need to hard-wire specific configurations. • *Extensibility* (**P3**) makes it easier to integrate diverse applications. • *Expressiveness* (**P4**) provides for the different coordination patterns that may be needed. • *Easy Debugging* (**P6**) makes it easier to debug user problems. • *Failure Tolerance and Recovery* (**P9**) minimizes the impact of failures on users. • *Application Portability* (**P10**) encourages development of a larger set of applications that can be composed.
H3. Human Level Performance Needs	• *Limited Temporal Decoupling* (**P1**) allows data to disappear after its period of relevance is exceeded. • *Perceptual Instantaneity* (**P7**) insures system is fast enough to provide for human performance needs. • *Scalability to Workspace-sized Traffic Loads* (**P8**) is limited by the traffic humans can generate.
Based on Technology Factors	
T1. Hardware Heterogeneity	• *Simple and Portable Client API* (**P5**) minimizes effort required to support new hardware platforms.
T2. Software and Software Platforms Heterogeneity	• *Expressiveness* (**P4**) provides a variety of coordination patterns that may be needed by legacy software. • *Simple and Portable Client API* (**P5**) minimizes effort required to support new software platforms.
T3. Short Timescale Change	• *Limited Temporal Decoupling* (**P1**) masks transient failure, and prevents system performance degradation by limiting build-up of tuples. • *Referential Decoupling* (**P2**) discourages application interdependence, and minimizes the chances of cascading failures. • *Failure Tolerance and Recovery* (**P9**) prevents crashes from causing systemic failure and allows for quick recovery.
T4. Long Timescale Change (Space evolution)	• *Extensibility* (**P3**) allows applications to be adapted as workspace evolves. • *Expressiveness* (**P4**) provides coordination patterns that may be needed by future applications. • *Simple and Portable Client API* (**P5**) minimizes effort required to support future platforms. • *Easy Debugging* (**P6**) makes it easier to troubleshoot integration of new technology. • *Application Portability* (**P10**) provides a broader selection of new applications with which to evolve workspace functionality.

Table 4 – Provision for Interactive Workspace Characteristics

4.2 Tuplespace Features and Needed Extensions

Given the properties just described in the previous section, the question becomes what features are necessary for a coordination infrastructure? We propose the Event Heap, an extended version of the tuplespace model. After describing in more detail the tuplespace model (discussed briefly in Chapter 2, Section 2.3), this section presents the design aspects of the tuplespace system model that give it some but not all of the properties that were advocated in the first half of this chapter, followed by the extensions needed to satisfy the remaining properties.

Also, recall that the contribution of this dissertation is not in the implementation of any given feature. With the exception of the automatic routing fields, all of the other features have been implemented in some other coordination system. The combination of features we describe is necessary for any interactive workspace coordination system, however, and these features have not been implemented together before.

Table 5 summarizes the system features, both intrinsic to tuplespaces and specific to our extensions, and shows how they relate to the desired system properties P1 through P10. While in most cases, no one feature completely provides for a given property, in aggregate the features in the table (and discussed in this section) provide for all of the previously defined properties. Also, recall that the contribution of this dissertation is the synthesis of a set of features appropriate for coordination in an interactive workspace, and not the implementation of any given feature. As such, with the exception of the automatic routing fields, all of the other features have been implemented in some other coordination system. The combination of features presented here, has not, however, been combined before in any one coordination infrastructure.

System Feature	System Property Provided or Aided
Routing	
F1. Content Based Addressing	• *Referential Decoupling* (**P2**). • *Expressiveness* (**P4**) by allowing flexible content selection.
F2. Support of All Routing Patterns	• *Expressiveness* (**P4**).
F3. Standard Routing Fields*	• *Extensibility* (**P3**) by encouraging routing compatibility between applications. • *Application Portability* (**P10**) by insuring that the same fields are used for routing in different interactive workspaces.

Continued on next page

* Not provided by basic tuplespace model.

Continued from previous page

System Feature	System Property Provided or Aided
Persistence	
F4. Limited Data Persistence*	• *Limited Temporal Decoupling* (**P1**). • *Failure Tolerance and Recovery* (**P9**) by preventing tuple buildup that could lead to system instability.
F5. Query Persistence/ Registration*	• *Extensibility* (**P3**) by making it possible to reliably snoop on communication. • *Easy Debugging* (**P6**) by enabling snooping to monitor application interactions. • *Expressiveness* (**P4**) by allowing guaranteed tuple receipt.
Communication Transparency	
F6. Transparent Communication	• *Extensibility* (**P3**) by enabling snooping, interposition and stream transformation. • *Easy Debugging* (**P6**) by enabling monitoring of application interactions.
F7. Self-describing Tuples*	• *Extensibility* (**P3**) by allowing existing applications to be more easily integrated with new ones. • *Easy Debugging* (**P6**) by making it easier for humans to understand inter-application communications. • *Application Portability* (**P10**) by allowing reverse engineering of applications to integrate them in new environments.
F8. Flexibly Typed Tuples*	• *Application Portability* (**P10**) by allowing additional fields to be added to tuples without breaking other applications. • *Extensibility* (**P3**) in a similar fashion to **P10**.
Distribution of Infrastructure	
F9. Logically Centralized	• *Limited Temporal Decoupling* (**P1**). • *Referential Decoupling* (**P2**). • Becomes feasible because *Scalability to Workspace-sized Traffic Loads* (**P8**) limits traffic to that which can be generated by devices in a single workspace.
F10. Physically Centralized	• *Simple and Portable Client API* (**P5**) by reducing client code overhead. • *Perceptual Instantaneity* (**P7**) relaxes performance constraints that might make the bottleneck of a centralized system intolerable. • Becomes feasible because *Scalability to Workspace-sized Traffic Loads* (**P8**) limits traffic to that which can be generated by devices in a single workspace.
Infrastructure API	
F11. Simple API	• *Simple and Portable Client API* (**P5**).
F12. General API	• *Extensibility* (**P3**) by permitting arbitrary coordination code. • *Expressiveness* (**P4**).
Ordering	
F13. At Most Once, Per Source FIFO*	• *Extensibility* (**P3**) by not requiring special application code for ordering.
Failure Tolerance	
F14. Modular Restartability*	• *Failure Tolerance and Recovery* (**P9**).

Table 5 - System Features and Related Properties

4.2.1 Traditional Tuplespaces

Tuplespaces is the name that Gelernter and Carriero coined in the mid-1980s for the coordination system instantiated by Linda [64]. The Linda system is a method for coordinating parallel programming, which is based on *generative communication.* This means that senders generate data for use by others without needing to know which process will consume it, and consumers of data need not know who created it. Data sent is in the form of tuples, where each tuple is an ordered set of typed values of some size. Senders place these tuples in a tuplespace, and consumers request tuples using a template tuple whose field values are specified where an exact match is desired and left as wild-cards where values are desired to be retrieved from the matching tuple. When a match is found, the tuplespace infrastructure returns the matching tuple to the requesting process. Linda has two key properties: *communication orthogonality* and *free naming*. The former means that processes can be decoupled in space and time, and the latter that the tuples have no name aside from field contents and order, so tuple creation and matching serves to identify and effect the transfer of tuples between processes. Carriero and Gelernter also argue that many other coordination techniques can be accomplished using Linda.

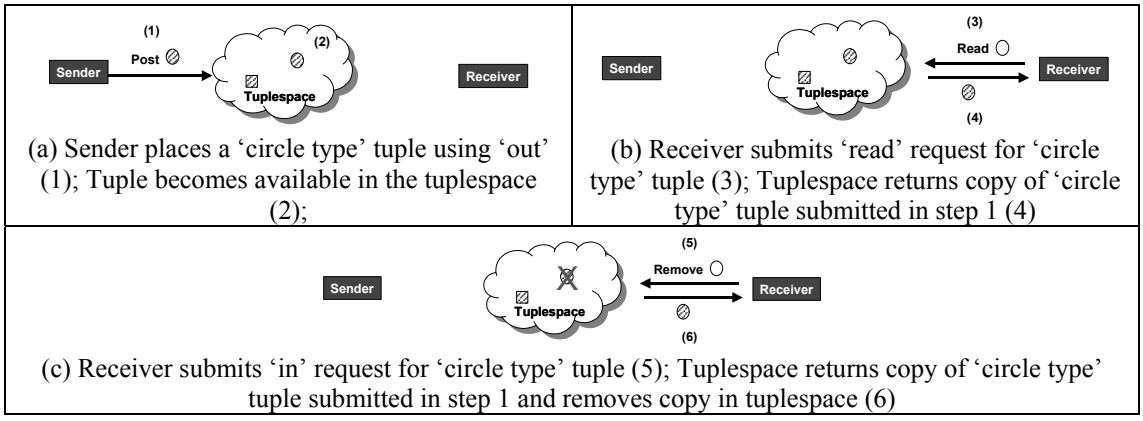


Figure 5 - Tuplespace System Diagram

The most important language primitives in tuplespaces are 'out' (puts a tuple into the space), 'in' (consume a tuple from the space), and 'read' (copy a tuple from the space), where the 'in' and 'read' operations supply a template tuple that may specify explicit values or wildcards for any tuple fields. Figure 5 shows an abstract representation of how tuplespaces function.

A variety of tuplespace derivatives have been introduced since Linda that modify the basic features, in some cases making them more compatible with the needs of interactive workspaces. There is, however, no canonical replacement to the original Linda tuplespaces model, so we use Linda for comparison. In Chapter 8, Section 8.3, we present related work on tuplespace derived coordination systems including some of the "modern" tuplespace systems. A specific comparison

of each of these modern systems is given there along with Table 18 which summarizes how their features align with the extensions we propose to the basic tuplespace model for the Event Heap.

4.2.2 Design Features of the Basic Tuplespace Model

Routing

Routing involves the aspects of a coordination system that determine how a message gets from a sender to a receiver. This involves addressing (how senders and receivers are determined), whether senders or receivers determine routing, what routing patterns are supported, and whether message transmission is push or pull based.

For addressing, sources and recipients may be specified either explicitly, or logically through a level of indirection. Tuplespaces provide logical routing through the use of content based addressing—message delivery is determined by the matching of attributes in tuple fields, with neither sender nor receiver specifying one another. We refer to content based addressing as feature **F1**. It provides referential decoupling, one of the needed system properties (**P2**). In addition, content based routing provides a more expressive way for receivers to choose tuples of interest since they can receive tuples based on arbitrary combinations of field values (supporting property **P4**). In tuplespace systems, content based routing is combined with receiver based routing, which means that receivers determine which content they will get. Note that content based routing may also be used with systems using sender based routing as is the case for the Intentional Naming System [19].

Tuplespaces also support all of the following routing patterns:

- **Unicast (Point-to-Point):** Send a message from a sender to a specific receiver. Accomplished in tuplespaces by having a receiver match for a specific value in a tuple field indicating that it is the recipient and then doing either a destructive ‘in’ call, or a non-destructive ‘read’ call.
- **Broadcast (One-to-all):** Send a message from a sender to all receivers. Accomplished in tuplespaces by having all receivers match on a special broadcast value in a standard tuple field and doing a non-destructive ‘read’ call.
- **Multicast (One-to-N):** Send a message from a sender to a group of receivers. Accomplished in tuplespaces by having all receivers in the group match on a special value representing the group in a standard tuple field and doing a non-destructive ‘read’ call.

- **Anycast (One-to-exactly-one-other):** Send a message from a sender to exactly one of a collection of possible receivers. This is useful for submitting messages that need to be processed exactly once by one of several valid receivers. Accomplished in tuplespaces by having all receivers in the group of valid receivers match on a special tuple field value representing the group, and doing a destructive 'in' call so that only the first recipient to match will see the tuple.

The ability to support all of the different routing patterns we call feature **F2**. This feature makes tuplespaces' API more expressive (property **P4**), allowing many types of coordination to be programmed in the system. It also allows coordination at the application level to mimic the patterns used among humans in CSCW work, which Johnson-Lenz claims in [89] are: one-to-one, one-to-many, many-to-many, many-to-one.

Persistence

Data persistence ensures that messages don't disappear immediately after their creation. This is provided by tuplespaces, since tuples are maintained in the tuplespace until a receiver performs the destructive removal operation 'in.' This full persistence provides temporal decoupling (**P1**) of processes or applications using the tuplespace, since the recipient need not be running at the time the tuple is created. We identified *limited* temporal decoupling (therefore limited persistence) as the desired property, however, and traditional tuplespaces provide no way of 'expiring' unconsumed tuples after their period of utility is over. We return to this as an extension later in Section 4.2.3. Still, together with this expiration feature, the system as a whole has *limited data persistence*, which we call feature **F4**.

Query persistence (**F5**) provides a similar guarantee for requests to receive messages. In receiver-routed systems, the recipient specifies messages to receive, in the case of tuplespaces by providing a template that a candidate tuple must match before it is received. If that 'query' is allowed to persist, the system can ensure that a receiver gets a copy of all messages that match the query. In tuplespaces, there is no query persistence—the template is passed as an argument to the retrieval call and is forgotten by the system as soon as the request is satisfied. This is known as a 'pull' or polling based system, while systems with query persistence are 'push' based systems. The absence of query persistence means that tuples that are placed and deleted between two polls from an application will not be seen by the polling application, making it difficult or impossible to write debugging (**P6**), logging and snooping (**P3**) applications. To allow for these important types of applications, tuplespaces must be extended to support query persistence, as will be discussed in Section 4.2.3.

As a nice side effect, allowing query persistence also reduces network overhead, since 'pull' based systems require a complete round-trip over the network for each tuple retrieved—the request must be sent to the server and the result returned. With query persistence, one request to the server suffices for the return of all future matching events until the query is removed.

Transparency vs. Opacity of Communication

Transparency of communication is the degree to which applications using the system infrastructure are able to observe and to some degree interpret communications among other entities. Tuplespaces are transparent since all posted tuples may be seen by all participating applications until they are removed. (Applications cannot determine, however, which applications receive copies of any given tuple). *Transparent communication* we refer to as feature **F6**. Providing transparency makes a system more extensible (**P3**) by allowing snooping and interposition (although to work consistently the query persistence issue mentioned under persistence must also be addressed), and makes it easier to debug (**P6**) the system by observing communications.

The richness of the data format and the opacity of the format (how easy it is to perform introspection on a message) also affect communication transparency. The tuplespace model provides a relatively simple data format: tuples contain an ordered set of fields, each with a primitive type and value—no nesting of tuples is permitted. Therefore, tuples with the same number of fields and field order but different semantic meanings for fields cannot be disambiguated (e.g. a tuple with a single integer field whose value specifies a page number is indistinguishable from one whose integer field specifies a file descriptor number). This is a drawback we will address in Section 4.2.3 with the flexible typing extension.

Format opacity refers to how difficult it is for a party that knows nothing about a message to determine its contents. A message format with destination and a payload of bytes is completely opaque, while one that provides information on semantic meaning in addition to the content is relatively transparent. Tuplespaces provide limited transparency: any application that retrieves a tuple may determine the number of fields, field types and field content, but the tuple as a whole is not typed or named, nor are individual fields named, so it is not straightforward to determine the meaning of the tuple or fields unless one is an intended recipient of a message. The self-describing tuples extension that will be described in Section 4.2.3 addresses this deficiency.

It should be noted that there are two problems here, whether an application can determine the meaning of a message of unknown format, and whether a developer can look at a captured

message and determine the meaning. Format transparency would make the latter possible, but without sophisticated artificial intelligence, the former would remain unlikely.

Distribution of Infrastructure

Another design decision is to what extent the infrastructure is centralized or decentralized. The tuplespace model is logically centralized, with a central space that is used to exchange tuples. This logical centralization, which we call feature **F9**, makes it easier to keep applications referentially and temporally decoupled (**P1** and **P2**) since the system can act as a proxy between senders and receivers. Logical centralization does, however, reduce the scalability of the system (Internet scale coordination through a logically centralized infrastructure would not be feasible), but due to **P8** we only need the system to scale enough to support the devices in a single workspace.

In most implementations (JavaSpaces [7] and T Spaces [145], for example), the system is also physically centralized with a server machine hosting the tuplespace. This is also the approach we have chosen with the Event Heap, and we refer to this *physical centralization* as feature **F10**. While physically centralizing limits total system throughput and the scalability of a system, with modern processors and networks the performance of such a centralized tuplespace system is more than adequate for the tens or hundreds of devices and applications that can be expected to be in use during group collaborations in an interactive workspace.

By physically centralizing the system, the implementation can also be concentrated in the server, making the client software simpler (**P5**). This makes it easier to port the client API to new platforms. Although physical centralization creates a single point of failure, several implementations, such as [66] and [37], show how to distribute the tuplespace across several machines, so a cluster could be used instead of a single server. Further, our use of the Event Heap's temporal decoupling combined with soft state to regenerate data across Event Heap server failures allows an interactive workspace to tolerate most transient failures in the centralized tuplespace even without a distributed implementation. Another concern is how much performance a single server can provide, but the number of transactions per second and the latency necessary are limited by both the perceptual instantaneity property (**P7**) and the workspace sized traffic loads property (**P8**).

Infrastructure API

The tuplespace model has a very simple API (**P5)** with six functions that are used to express the coordination ('in', 'out' and 'read'—the three mentioned in Section 4.2.1—non-blocking versions

of the two read operators, and an 'eval' function which can be used to launch new processes). This makes it easy to port the system to new platforms. In addition, a simple API makes it simpler to add wrappers to existing application's programmatic interfaces when source code is unavailable (similar to "puppeteering" [53]), a critical ability in integration of legacy applications. While a simple API doesn't necessarily imply a simple implementation, for tuplespaces the client-side implementation can be made quite simple. The *simple API* feature we call **F11**.

Another important design characteristic of the API is its generality. Gelernter and Carriero [65] suggest that coordination languages (what we have been calling the API of our coordination infrastructure) should be thought of as orthogonal to computational languages. Each coordination language provides a set of primitives that can be used in any computation language. They further state that some coordination languages are 'general purpose' and can be used to express any type of coordination (analogous to a computational language being Turing-complete).

A main feature of the tuplespace model, they argue, is that it is a general-purpose coordination language (the generality of the API we call feature **F12**). As one example, RMI can be implemented with a set of two tuplespace calls on the calling application and two on the application receiving the call. [64] and [66] give examples of several other types of coordination that can also be accomplished with tuplespace semantics. This generality provides for portability to new platforms and for heterogeneity, both goals that we specified earlier for coordination infrastructures in interactive workspaces. In terms of our systems properties, it also means that tuplespace systems are more extensible (**P3)** and expressive (**P4)** since any type of coordination can be expressed using the tuplespace API.

4.2.3 Needed Extensions

The tuplespace model satisfies many of the desired properties for a coordination infrastructure for an interactive workspace, with notable shortcomings including application portability (**P10**) and extensibility (**P3**). The reasons for these stems from tuplespaces original intended use as a coordination system for parallel applications, in which all the processes were designed together. In these situations, there is never any coordination among processes that are foreign to one another. This issue has been noted in other work that strives to use tuplespaces in an open manner among applications not originally designed to work together [44].

The remainder of this section details a set of extensions to the tuplespace model to address its deficits in the interactive workspace domain. We call tuplespaces plus these extensions the Event Heap model. The model has been implemented in a system with the same name [84], which is

discussed in detail in Chapter 5. Tuples used by the Event Heap are slightly different from those in the basic tuplespace model. They are typed and have certain standard fields, among other things. To distinguish them from basic Linda-style tuples, we refer to them as events throughout the remainder of the dissertation. Other extended tuplespaces systems that have implemented some of these features are discussed in the last part of this section.

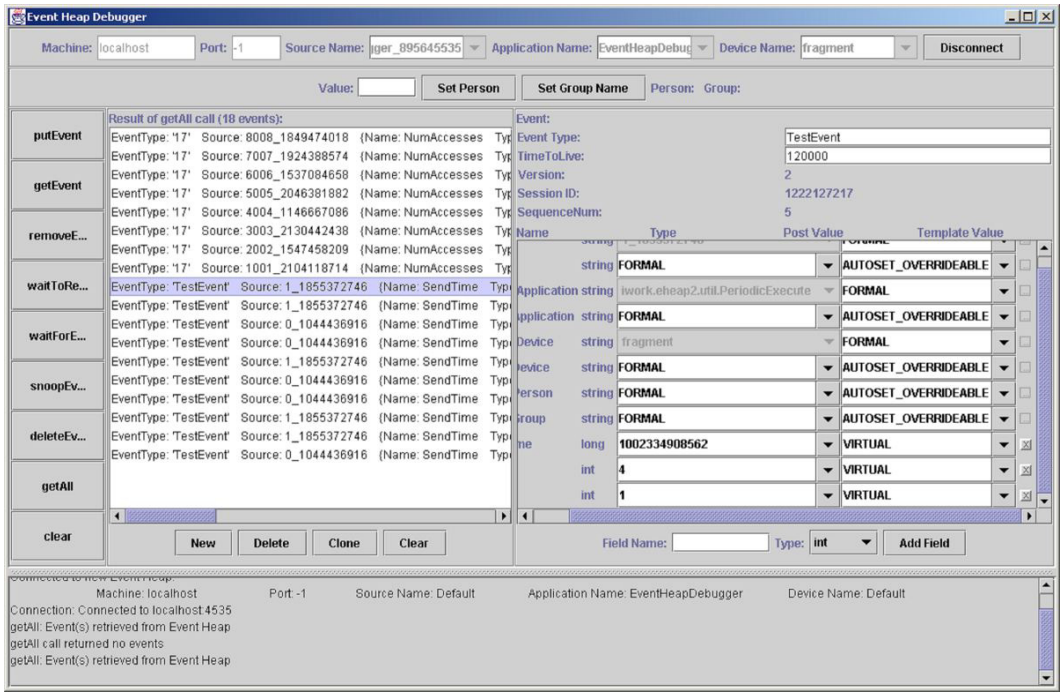


Figure 6 – Viewing Self-describing Tuples in the Event Heap Debugger

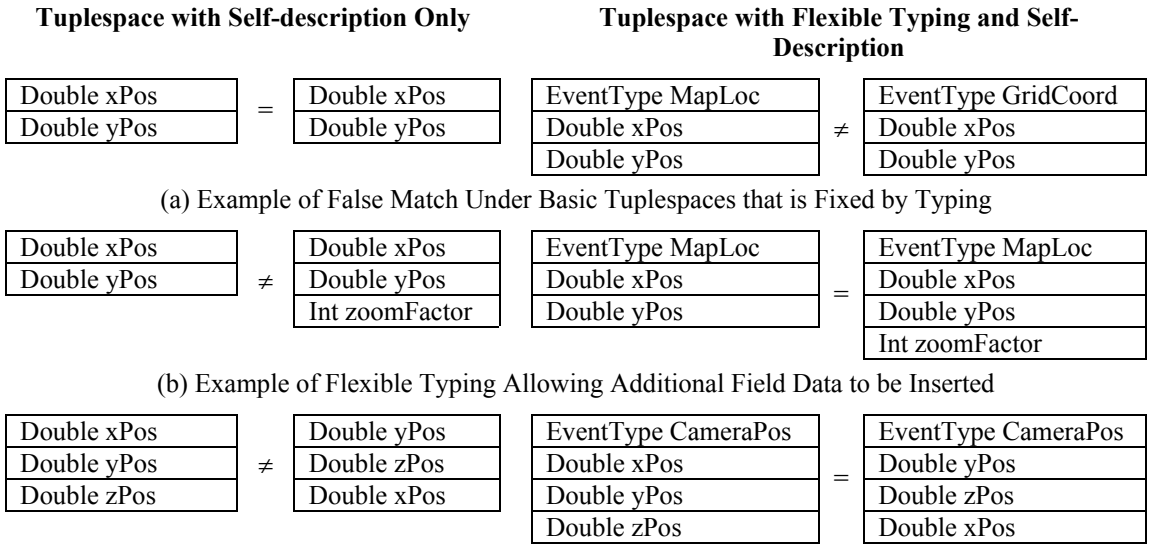
Self-describing Tuples

One problem with using tuplespaces in an interactive workspace is that the semantic meaning of a field in a tuple is known only to the programmer that programmed the processes using the tuplespace. This makes it difficult to reverse engineer tuple communication to integrate new applications with an existing collection of cooperating applications. This problem can be avoided by adding a string containing a name to the type and value for each field in order to make the tuple fields self-describing. We call this feature **F7**[3]. With self-description, a field can be called, for example, 'Xpos' if it contains the x position for the data being represented by the tuple. Combined with tuple typing, the extension that will be discussed next, self-description makes it more likely that new developers extending an existing application can infer the meaning of the tuple and write applications to emit and react to the tuple (note that it most likely won't help applications at run-time since parsing text and inferring meaning is a challenging AI problem).

[3] Since it contributes to communication transparency (**F6**), this feature is given a number in that grouping.

This provides for additional extensibility (**P3**) and improves application portability (**P10**). Finally, it also makes it easier to debug the system by simplifying the monitoring of traffic in the tuplespace (**P6**). Self-describing fields have been used in other extended tuplespace implementations including TSpaces [145], and they have also been used in publish-subscribe systems such as the Information Bus system [115].

In our project, self-describing fields have proven most useful in debugging and adapting applications. We have written a debugger application (Figure 6), which allows users to browse the current state of the Event Heap and perform the basic operations through a user interface. Since event fields have human readable names, it is relatively easy to look at the events and determine their function. One can look at the Event Heap in the debugger and determine whether an action didn't complete because no event was generated, or because the event did not reach a valid target. New graduate students on the project have also been able to adapt their Event Heap code to work with old code by looking at the events generated by the old application. They then create "clones" of the events and test out how changing field values and submitting the events affects the behavior of the old application.



(a) Example of False Match Under Basic Tuplespaces that is Fixed by Typing

(b) Example of Flexible Typing Allowing Additional Field Data to be Inserted

(c) Example of Field Ordering Causing Mismatch which is Fixed by Flexible Typing

Figure 7 - Flexible Typing Examples
('=' : Tuples will match if values match; '≠' : Tuples cannot match)

Flexible Typing

In a parallel application, the programmer or programmers can determine standard formats and meanings for the tuples that will be exchanged ahead of time. In an interactive workspace, applications developed separately might choose tuples of the same size and field order, but with

different semantics, causing erratic behavior when collisions occur. These problems can be solved by the introduction of flexible typing, which we refer to as feature **F8**[4]. Figure 7 shows some of the problems that can arise by showing the differences between matching in a basic tuplespace extended only with self-description, and matching in a tuplespace with both flexible typing and self-description.

Typing is accomplished by adding a special tuple type field (or event type in the case of the Event Heap) whose value determines the minimal set of fields required for this tuple type and the meanings of each of these required fields. With typing, applications need only avoid collisions on the name of this type, which solves the collision problem for the majority of the cases and provides for application portability to new spaces (**P10**). The type of problem fixed by this is shown in Figure 7a. Typing has been implemented in other extended tuplespace systems such as JavaSpaces [7], and L2imbo [52].

Note that the issue of typing and the issue of naming are inter-related when tuplespace-based coordination is used. Typically, naming is used to specify targets for messages, and types are used to specify content of messages. Since tuplespaces use content-based routing, the type of the message typically plays a role in determining which recipients will receive any given tuple (i.e. those recipients that are matching on the given tuple-type). Thus, providing a tuple type field in some measure both prevents type collision and provides a means of naming intended recipients. More information on naming, typing, and storage and transmission of objects can be found in [15], among other places. The issue of typing and naming is also further discussed in this dissertation in Chapter 9, Section 9.1.

Besides avoiding semantic collisions, there is the problem of allowing newer applications to use an enhanced version of a tuple while maintaining interoperability with older applications. This problem can be avoided by making tuple typing 'flexible.' Specifically, matching can be changed to ignore field order and allow matching to any tuple that has a superset of the template's fields (as in Figure 7b). Thus, newer applications can add fields with supplemental information without breaking compatibility (analogous to adding experimental headers in HTTP or extra headers to email as specified in RFC-822) making the system extensible (**P3**). Flexible typing has been advocated for software systems in general in [133]. It should also be noted that having a flexible data format is relatively independent of choosing to use an extended tuplespace for coordination. For example, we expect that similar flexible typing could be applied in message passing or publish-subscribe coordination systems.

[4] Along with self-description, this feature is numbered in the communication transparency group.

We have needed the flexible typing feature on several occasions during development of applications for our project. The multibrowsing system [88] allows users to push and pull web pages between computers in the iRoom. At one point we upgraded the Windows version of the system to support specification of whether the newly retrieved page was to be opened minimized, or maximized, and whether it should be brought to the front. We did this by keeping the same event format while adding on extra fields for the Windows version. Making these changes didn't even require a recompile of the Linux version of the application, which was able to ignore the new fields.

Standard Routing Fields

While the standard tuplespace model supports all of the routing patterns (one-to-one, broadcast, etc.), individual application developers must have a convention for which tuple fields are set to determine their routing. This works well for parallel applications with processes designed to work together. In the 'open' tuplespace environment of an interactive workspace, however, a standard for fields is needed to insure a compatible routing mechanism between all applications. This in turn enhances the extensibility of the system (**P3**) and provides better application portability (**P10**). We call the addition of standard routing fields feature **F3**[5].

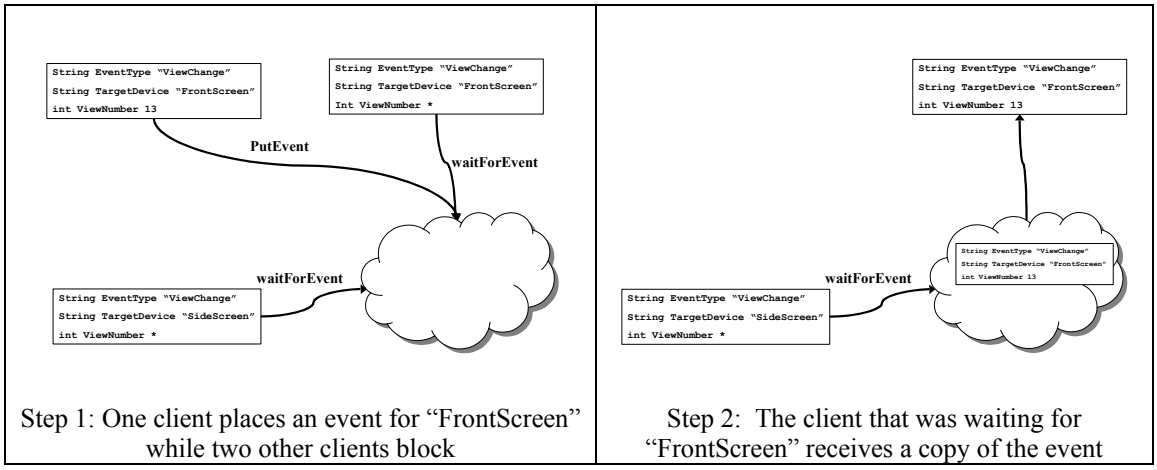
With the standard routing fields extension, the Event Heap client implementation tags the source fields of posted tuples with information about the source and the target fields of template tuples with information about the querying receiver. By default, the target fields are set to wildcards on senders, allowing the event to be routed to any receiver, and source fields are set to wildcards on receivers so they match events from any source. By overriding the target values of an event sent from a source, an application programmer can route the event to a specific target instead of all targets, and by overriding the source values used by a receiver the programmer can select only events from a specific source rather than events from all sources.

As an example, to send to all '3DViewer' applications, a sender can set the 'TargetApplication' field on an event to '3DViewer.' Since the client-side of the infrastructure sets the 'TargetApplication' field of template events on all 3D Viewer applications to '3DViewer,' the event will only be picked up by 3D Viewer applications (although only by those that would otherwise match the event which was sent). If an event is sent without the application writer overriding the value of 'TargetApplication' field, that field will be sent out set to a wildcard value

[5] This features number appears out of order, but is actually correct since it falls in the category of routing along with content-based addressing and support for all routing patterns (**F1** and **F2**, respectively).

and will therefore match all receivers looking for an event that otherwise matches the one being sent.

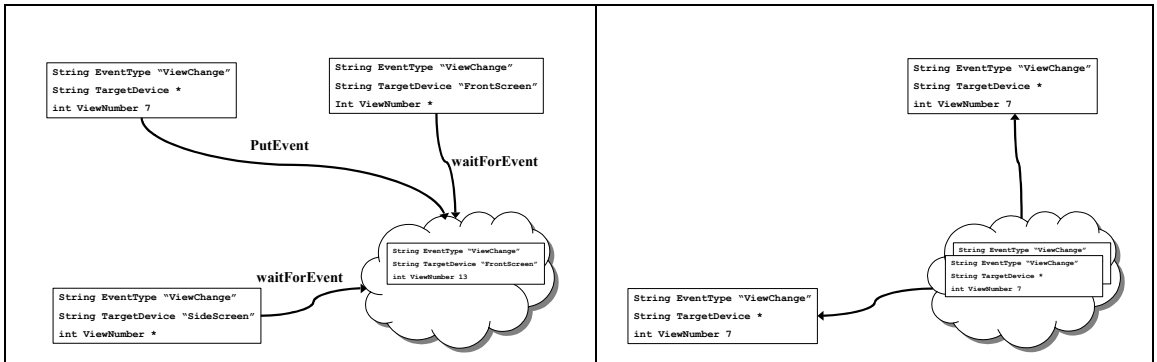
The Event Heap supports routing by program instance, application name, device, person, group or any combination of these. The types of routing to support were determined by consulting with several research groups, including a design research group in the Mechanical Engineering Department, that are using the Event Heap to write applications for their specific research areas. Figure 8 gives another example of using the routing fields to either send to a specific screen, or multicast to all screens.



Step 1: One client places an event for "FrontScreen" while two other clients block

Step 2: The client that was waiting for "FrontScreen" receives a copy of the event

A source sends an event to the "FrontScreen".



Step 1: One client places an event for target device '*' while two other clients block

Step 2: Both blocking clients match on target device '*' and receive a copy of the event

Later, a source sends an event to all screens.

Figure 8 - Using the Event Heap Standard Routing Fields

We do not yet have a lot of experience using the default routing fields since they were only added in the most recent Event Heap release. Nonetheless, they were implemented to address a real problem we were seeing with applications in the iRoom. Specifically, a common technique was to generate an event to trigger an action on some specific touch screen in the room. This was

done by creating an event field indicating the target, and having each potential target wait for events with its ID in the target field. Unfortunately, since developers all used different field names and ID schemes, every time different applications needed to be integrated the developers of all components needed to be gathered together, or careful sleuth-work with the Event Heap debugger had to be done. Self-description made it relatively easy to determine which fields contained the target, but since developers were using integer IDs for the value of the target, it was difficult to determine the meaning of each ID. We are hopeful that having a standardized method for routing will eliminate this problem in the future.

Tuple Expiration

The standard tuplespace model provides temporal decoupling, but not the limited temporal decoupling we had specified as a desirable system property. In an interactive workspace there is no guarantee that a tuple posted by an application will ever be consumed (perhaps the intended recipient has crashed or is not functioning), so if tuples aren't expired they can build up in the tuplespace, leading to the eventual exhaustion of server resources and crash of the server software. This leads the system to be more failure prone, the opposite of what is specified by property **P9**.

We extend tuplespaces by adding an expiration field, which is set by application developers, to all tuples. This gives the system limited temporal decoupling (**P1**). It also allows each tuple to have an expiration period proportional to the time over which it would appear reasonable to users of the room to see a causal effect of that tuple (as determined by the developer). Tuple expiration along with the data persistence of the standard tuplespace model provides the limited data persistence feature (**F4**). Other extended tuplespace implementations, including T Spaces [145] [99] have expiration for similar reasons.

Query Persistence/Registration

Another drawback of the basic tuplespace model is that it only supports polling. This means that tuples placed into the tuplespace and removed between successive polls by a process will not be seen by that process. In the controlled environment of a parallel application using a tuplespace, this race condition can be avoided by careful programming, but in an interactive workspace with a diverse collection of applications it cannot be. Further, the introduction of the expiration extension makes the problem worse since tuples may simply expire before they can be retrieved by a polling process.

The solution is to add an extension that allows a query to be registered with the coordination system. The query persists until it is deregistered, and for as long as it persists, a copy of each posted tuple which matches the registered template is returned to the querying application by a notification call. This is, in fact, the only mode of retrieving messages in a publish-subscribe system.

Query registration and persistence make the system easier to debug (**P6**) since trace applications can log all tuples placed into the tuplespace. It also improves extensibility (**P3**) since it allows snooping applications (see Extensibility in Section 4.1.3) to spy on and react to communications even between applications that perform destructive reads of tuples. The overall expressiveness of the system (**P4**) is also improved since this type of guaranteed receipt is not possible with standard tuplespaces. We refer to *query persistence/registration* as feature **F5**[6]. Query registration has been a popular feature in other extended tuplespace implementations including JavaSpaces [7], T Spaces [145] and Lime [111].

Query registration was added to the Event Heap specifically to support logging applications when we realized that there was no other way to insure receipt of all events during the time an application was connected to the Event Heap.

FIFO, At Most Once Ordering

Normal tuplespaces provide no guarantee of the order in which tuples are returned by a series of identical queries. This has a strong negative impact on applications for an interactive workspace. In practice, most applications perform a loop retrieving tuples that match some particular schema, and then performing some action based on the contents of the retrieved tuple. To allow other applications to react to the same tuple (which in turn permits multi-cast routing), applications by default perform a non-destructive read. On the next call in the loop, however, the same tuple may be retrieved again since it will still be a valid match to the template. In order to not react twice to the same tuple, code must be written to track which tuples the application seen, which adds development overhead. The problem of duplicate retrieval is known as the multiple read problem, and has been noted by [129]. The additional development time could discourage developers from doing it the right way and lead them to use destructive reads instead (we saw developers do this in early Event Heap versions that didn't support ordering).

[6] This is considered feature number five since it goes in the persistence category along with feature four, data persistence.

Having applications perform destructive reads reduces extensibility of the system, which detracts from **P3**. Since most applications use this feature, and we want to maintain extensibility, it is desirable to add *per-source FIFO, at most once, ordering,* or some stricter ordering, to any extended tuplespace system intended for an interactive workspace. We call this feature **F13**, and adding it ensures that each tuple will be seen at most once, and the oldest unseen tuple from a source will always be returned before newer unseen tuples. While total ordering would allow applications to perform more sophisticated reasoning on tuples read, having it as a rigid specification would make it more difficult to implement clustered versions of the Event Heap model[7]. Further, Oki found in [115] that exactly once, in order, was sufficient for most applications. For the Event Heap, exactly once delivery is only provided if query registration is used, otherwise, since events may expire before they are retrieved by a given application, only at most once delivery can be provided. The Event Heap implementation does in fact provide total ordering[8] to most applications as a side effect of having a physically centralized server-based implementation. Per-source ordering is also in LIME [111] and has been found useful when applying the tuplespace model to other domains (see, for example, [100]). To our knowledge, no tuplespace system or application of tuplespace systems has needed stronger than per-source ordering (such as total causal ordering).

It became clear very early on in the development of the Event Heap that providing FIFO, at most once ordering, was important. During the design of the projector control system for the iRoom, we needed a means of telling projectors to turn on or to display different video sources. Unfortunately, using basic tuplespace semantics through T Spaces (which we were then using as the underlying system for the Event Heap), the same projector event was picked up repeatedly until it expired. While we initially tried several ad-hoc indexing schemes just for that application, as more programs were developed it became clear that this was a common problem and we added FIFO, at most once ordering into the infrastructure.

Beyond basic ordering and delivery, some systems provide more advanced delivery semantics such as transactions, guaranteed delivery and consistent views of data. We believe as Oki [115] does that most applications will not need this functionality and that the performance penalty incurred by providing them in the coordination system is not worthwhile. For those applications

[7] While Lamport showed in a famous paper [96] that a distributed system could provide consistent total ordering, his solution does not guarantee wall-clock ordering. In an interactive workspace, face-to-face communication provides an independent channel that makes it possible for users to see the results of triggered actions come out of order. This could be a disconcerting experience for users.

[8] While we have not proven this, an informal analysis by the reader of the implementation described in Chapter 5 should convince them that this is the case.

that require these features, a higher level API could be built on top of the Event Heap. More information on transactions can be found in [28].

Modular Restartability

In addition to the basic failure tolerance provided by tuplespaces through decoupling, an important extension for the interactive workspace domain is modular restartability (feature **F14**). Modular restartability implies that any application or even a component of the central infrastructure can be restarted without causing failure in other components. In the Event Heap implementation, the client side API is designed to automatically reconnect should the server go down and later restart. The server itself was, of course, also programmed such that crashing clients do not affect it. During development this has allowed us to restart client applications at will, restart the server after a crash, or bring up a new version of the server software without having to restart client applications running on the various machines in the iRoom. Modular restartability was not included in the original tuplespace model since in most parallel applications all processes need to run to completion to solve the problem. Thus, failure in one always required a restart of the whole system. Modular restartability helps support property **P9**.

4.2.4 Event Heap Feature Summary

Table 6 shows how the Event Heap features, which come from those already present in the tuplespace model and the extensions just described, provide the Event Heap model with the properties, which were described in Section 4.1, that are needed for application coordination in an interactive workspace. This table provides the inverse mapping of the one given in Table 5.

4.3 A Note on Performance

One of the characteristics we specified for interactive workspaces was that the software infrastructure would be bound by human performance needs (**H3**), and the associated property, **P7**, stated that the coordination system must have the property of supporting perceptual instantaneity. Performance is not a feature designed into the system, but rather a characteristic determined by features of the system and how well they have been implemented. No matter how well the features we specify provide the other system properties, if it is not possible to implement the system such that perceptual instantaneity is achieved, the overall model is not tenable.

System Property		Features that Help Provide Property
P1.	Limited Temporal Decoupling	• *Limited Data Persistence* (**F4***). • *Logical Centralization* (**F9**) provides central location to buffer data when neither coordinating application is running.
P2.	Referential Decoupling	• *Content Based Addressing* (**F1**) provides routing mechanism that doesn't require coupling. • *Logical Centralization* (**F9**) provides an intermediary to decouple coordinating applications.
P3.	Extensibility	• *Standard Routing Fields* (**F3***) encourages routing compatibility between applications. • *Query Persistence/ Registration* (**F5***) makes it possible to reliably snoop on communication. • *Transparent Communication* (**F6**) enables snooping, interposition, and other types of stream transformation. • *Self-describing Tuples* (**F7***) allow existing applications to be more easily integrated with new ones. • *Flexibly Typed Tuples* (**F8***) allow additional fields to be added to tuples without breaking other applications. • *General API* (**F12**) permits arbitrary coordination code. • *At Most Once, Per Source FIFO Ordering*(**F13***) insures the same technique for ordering is used for all applications.
P4.	Expressiveness	• *Content Based Addressing* (**F1**) allows flexible content selection. • *Support of All Routing Patterns* (**F2**). • *Query Persistence/ Registration* (**F5***) allows receipt of all tuples during registration period. • *General API* (**F12**) allows expression of any type of coordination.
P5.	Simple and Portable Client API	• *Physical Centralization* (**F10**) reduces the amount of infrastructure code on clients. • *Simple API* (**F11**).
P6.	Easy Debugging	• *Query Persistence/ Registration* (**F5***) enables snooping to monitor and debug application interactions. • *Transparent Communication* (**F6**) enables monitoring and debugging of application interactions. • *Self-describing Tuples* (**F7***) make it easier for humans to understand inter-application communications.
P7.	Perceptual Instantaneity	• Makes performance constraints achievable using a *Physically Centralized* (**F10**) implementation.
P8.	Scalability to Workspace-sized Traffic Loads	• Limits traffic which needs to be handled by *Logically Centralized* (**F9**) and *Physically Centralized* (**F10**)implementation.
P9.	Failure Tolerance and Recovery	• *Limited Data Persistence* (**F4***) prevents tuple buildup that could lead to system instability. • *Modular Restartability* (**F14***) insures that transient server failures or other component failures don't impact the system as a whole.
P10.	Application Portability	• *Standard Routing Fields* (**F3***) insure that the same fields are used for routing in different interactive workspaces. • *Self-describing Tuples* (**F7***) allow easier reverse engineering for integration of applications in new environments. • *Flexibly Typed Tuples* (**F8***) allow additional fields to be added to tuples without breaking older applications.

Table 6 – Provision for Event Heap Model Properties

* Not provided by basic tuplespace model.

Our verification that satisfactory performance can be achieved is through our prototype implementation of the Event Heap model. Tests of the system show that it can handle several hundred connected devices, and several hundred events per second at latencies of around 50 ms for a round trip (post to the Event Heap followed by a retrieval of the same event). Our code is not well optimized, but nonetheless the system's performance exceeds the needed 100 ms of latency, even under the conditions we expect from an interactive workspace sized traffic load (**P8**). A more detailed presentation of the measurements can be found in Chapter 6, Section 6.3.

4.4 Facilitating Dynamic Application Coordination

One of the important usage capabilities for interactive workspaces that we identified in Chapter 3, Section 3.4 was dynamic application coordination. The Event Heap model provides for dynamic application coordination in the following way: applications listen for events to which they know how to respond, and emit events as users interact. Since the applications don't listen for events from any specific source, nor do they send out events to a specific destination, the applications interact which whichever other applications are currently running in the interactive workspace and using the same set of events. In our previous example of calendar applications, each would post date change events whenever users changed the date within that application and listen for date change events from other applications. This type of interaction is analogous to how humans interact within a room—even when divided into sub-groups, all conversation is audible, and any given person can choose to respond to something they hear. Thus, even though no formal interaction mechanism has been defined for the people in the room, they are able to coordinate their activities with one another. Similarly, applications "speaking" the same event types can coordinate with one another even though no specific mappings between the applications has been established.

4.5 Other Alternatives

While in the end we determined tuplespaces to be the coordination infrastructure best suited to interactive workspaces, there are many other coordination infrastructures that were candidates. Table 7 compares the tuplespace model and three other systems. The Event Heap extended tuplespace model is not included in the table since it has all of the properties by design (it would be a column of check marks).

The remainder of the section describes RMI/RPC, Publish-Subscribe and Message-Oriented-Middleware (MOM) and how they fall short of the desired properties in more detail . Note that any of these coordination models could be modified and extended to provide the same set of

properties as the Event Heap, but tuplespaces required the least amount of change. As the table shows, publish-subscribe would have also been a reasonable candidate, but it had limited support for two of the properties (limited temporal decoupling and expressiveness), while tuplespaces only had limited support for one of the properties (application portability).

	Tuplespaces	RMI/RPC w/Rendezvous	Pub-Sub	MOM*
P1. Limited Temporal Decoupling	~	×	×	✓
P2. Referential Decoupling	✓	×	✓	~
P3. Extensibility	~	×	~	✓
P4. Expressiveness	~	×	×	×
P5. Simple and Portable Client API	✓	×	✓	~
P6. Easy Debugging	~	×	~	~
P7. Perceptual Instantaneity	✓	✓	✓	×
P8. Scalability to Workspace-sized Traffic Loads	✓	✓	✓	✓
P9. Failure Tolerance and Recovery	~	×	✓	✓
P10. Application Portability	×	×	✓	✓

Table 7 – Comparison of Coordination System Properties
(✓ = fully meets property, × = doesn't have property, or limited, ~ = mostly meets property)

4.5.1 RMI/RPC Systems with Rendezvous

RMI/RPC with rendezvous is one major category of coordination infrastructures. Remote Procedure Call (RPC) was one of the earliest methods for allowing applications to interact with each other across devices . The basic technique is to allow programmers to make calls to procedures on other machines as if they were local. The remote procedure call infrastructure handles marshaling procedure arguments, moving them to the remote machine, and using them to invoke against the appropriate procedure. The result of the remote call is then marshaled, moved

* As noted in the text, MOM systems vary quite a bit in terms of features. We have done our best to represent here what is most common.

back to the calling device and returned to the application. As object oriented programming became more popular, the technique was extended to become Remote Method Invocation (RMI), which allowed methods to be invoked on objects residing on other devices. At the infrastructure level, the basic mechanism remains the same.

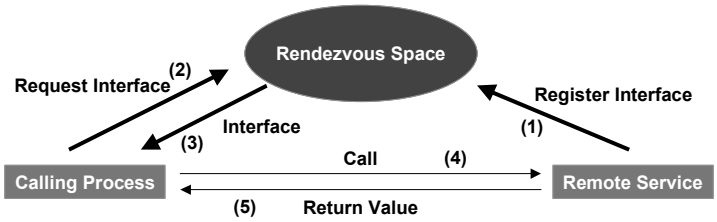


Figure 9 - RMI-Rendezvous System Diagram

While RMI/RPC works well if applications know ahead of time which devices will be running the collaborating applications, it does not work as well in a ubiquitous computing environment where spontaneous interaction is desired and the collection of interacting devices is constantly changing. To solve this problem, a rendezvous mechanism is typically introduced that allows applications to find other programs with which to interact using RMI/RPC. The systems typically work by having a well-known device that acts as a server and to which services can be registered. Applications that want to be made available for interaction register a resource-locator on the server, and programs that wish to interact can contact the server for a list of applications matching their specifications. Once an application has retrieved the resource locator, it can use it to retrieve the set of remote procedure or object handles for the remote application and begin making invocations against them.

Figure 9 shows an abstract RMI-Rendezvous system with the main features of actual systems. It also shows the typical sequence of steps required for coordination in such a system. First, a service registers its interface with the rendezvous infrastructure (1). A calling process requests an interface that matches its desired parameters from the rendezvous infrastructure (2), which then returns an object representing the remote service (3). Then the calling process makes calls to the representative object, which the system forwards, to the remote service (4). Finally, the remote service generates the result, which is passed back to the calling process (5). Steps 4 and 5 may then be repeated indefinitely for the same or different method calls on the reference object.

As an interactive workspace coordination infrastructure, RMI/RPC with a rendezvous system has many drawbacks. First, since the system is synchronous and transient, it is fundamentally coupled both referentially and temporally. This tight coupling also means applications are

interdependent—if the remote application hangs in a call, the calling application will also hang. This makes RMI/RPC systems less tolerant of transient failure. Further, as Banavar points out in [27], synchronous operation means clients need to stay connected for continued operation, which is unsuitable for domains like interactive workspaces, where clients may disconnect at any time.

Since both RMI and RPC mimic a function call, all communication is transient and synchronous, with the calling application blocking until the result is returned. This is a chief benefit of RMI/RPC systems since remote calls are made to look like standard local calls. This allows developers to program in a style with which they are comfortable, as if their applications were not distributed across different machines. This advantage is also a potential problem when developing using RMI/RPC since programmers may not realize that they need to account for potential problems associated with distribution (as noted in [61], and [91], among other places). One such problem is handling a connection loss to the remote device that occurs during an RPC call. Another is that RPC calls take much longer than local calls, and the caller is blocked while waiting for returned results. This can lead to potentially poor program performance, and wasting of processor cycles.

The next drawback of RMI/RPC is that it is relatively hard to extend collections of applications since the method invocation schemata and semantics must be known in order to integrate a new application—this also makes applications less portable for integration in new environments. This drawback is noted in [27], [107], and [61], among other places. In the last of these, Eugster notes that "individual *point-to-point* and *synchronous* communications lead to rigid and static applications, and make the development of dynamic large scale applications cumbersome."

The APIs of most RPC based systems are usually not portable since they are either tied to a specific language or include a great deal of client API overhead to allow for translation of calls and passed parameters between languages.

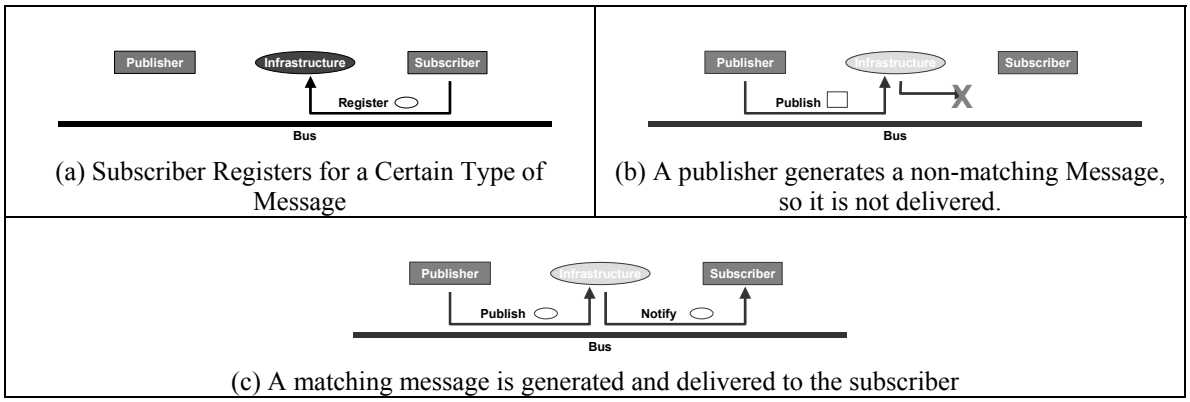
Debugging sets of applications constructed using RMI/RPC frameworks is also problematic. All communication is direct and opaque so it is difficult or impossible to trace inter-application communications for debugging purposes. As is pointed out in [27], opacity of communication also makes the system less extensible since it is not possible to perform interposition without modifying existing applications.

A final drawback stems from the nature of RPC: it is point-to-point so that is the only routing pattern that can be supported without significant development effort at the application level.

CORBA [146], Jini [141], HAVi [98], and UPnP [63] are examples of this type of infrastructure, although to be fair, some of them include extensions that allow them to avoid some of the drawbacks of a basic RMI/RPC system with rendezvous. These and other RMI/RPC systems are discussed in 0.

4.5.2 Publish-Subscribe Systems

Publish-subscribe systems are another major category of coordination infrastructures and provide a decoupled mechanism for distribution of information across devices. For such systems, there are two types of participants, publishers and subscribers, and any given application can take on one or both of the roles. Publishers broadcast messages to all other devices and applications connected to the system, and subscribers receive copies of all broadcast messages that conform to their specification. For example, a clock might send out messages at regular intervals that could be received by all applications wanting time information. Since sending applications can't determine who may receive a published message, and receiving applications need not know the source of a message to receive it, senders and receivers are said to be *decoupled*. To provide this decoupling, there is always, at least conceptually, a middleman that matches published messages to subscribers and insures the message is received by the appropriate applications. This can be accomplished by using a broadcast or multi-cast protocol on the local network and running infrastructure software on each device, so there need not necessarily be a server machine. Figure 10 shows how the basic publish-subscribe mechanism works.



(a) Subscriber Registers for a Certain Type of Message

(b) A publisher generates a non-matching Message, so it is not delivered.

(c) A matching message is generated and delivered to the subscriber

Figure 10 - Publish-Subscribe System Diagram

The ways of handling subscription in publish-subscribe systems vary in how flexibly applications can specify desired messages. In general, there is a tradeoff between overhead and expressiveness when using these techniques. Ones that allow more flexibility in choosing messages to be retrieved tend to have poorer performance or scalability. Eugster lists the following options in [61]:

- **Subject based:** Subscription based on a string (possibly hierarchical). This is good for platform interoperability, but is not very flexible.
- **Content-based:** A full set of attributes, a template object, or an executable filter are used to select messages to receive. More flexible but has higher overhead.
- **Type-based:** Messages have a type and can be sub-types. This minimizes the effect of name collisions on fields, potentially allows transparent use of serializable objects at the expense of platform independence.

A related type of coordination mechanism can be found in many shared object systems. In these systems, some of the objects are made to appear as if they are local to all machines, allowing any participating applications to read and write to them. Participating applications may also register and receive a notification message whenever member values of interest are updated by other applications in the ensemble. These systems are a hybrid or RMI and publish-subscribe systems and are therefore not treated separately.

After tuplespaces, publish-subscribe systems come closest to having the properties needed for coordination in an interactive workspace. However, they provide no temporal decoupling—an application must be running and subscribed at the time of message generation to receive a copy of the message. Further, publish-subscribe is not a general-purpose coordination system since it is designed primarily for broadcast and multicast. This makes other routing patterns and coordination types difficult or impossible; for example, anycast is not possible since there is no way to squelch a message once one of the receivers acknowledges receipt (in contrast, even the basic tuplespace model can support anycast if candidate recipients perform destructive reads to prevent others from receiving the same tuple).

Perhaps the best example of a publish-subscribe system is the Information Bus [115], which is discussed and compared to the Event Heap in 0. Eugster et al. give a good overview of publish subscribe systems and their properties in [61]. In addition, the SIENA system has shown how publish-subscribe can be extended to Internet-scales while keeping performance reasonable [38].

4.5.3 Message-Oriented-Middleware (MOM)

Message oriented middleware (MOM) is a hybrid of publish-subscribe and message-queuing with some enhanced message querying semantics. MOM coordination systems are designed to tie together sets of distinct business applications. As Banavar puts it, the goal of MOM is to provide "a technology that allows companies to automate such [business] processes by tying together its

organizations' independent applications" [27]. The characteristics of such ensembles include a need for loose coupling to provide independent evolvability, and the ability to tie in the legacy applications with minimal or no changes to the applications. Since MOM systems generally tie together groups of business computers, they tend to emphasize transactions, guaranteed delivery, and return receipts, and thus are relatively heavyweight. While some of the MOM goals of allowing integration of diverse systems are similar to those for interactive workspace infrastructure systems, their heavy-weight nature leads them to have higher latencies which makes it harder for such systems to provide fluid user interaction within an interactive workspace.

In more detail, MOM works in the following way: Applications may emit messages, and other applications may subscribe to messages. The continually generated stream of messages being either sent or received are known as message flows. Standard message formats are used by all participants. Receivers use a flexible query language to specify which messages they wish to receive, and all message content can be used in that specification. The system takes care of queuing messages for clients that go down and assuring delivery. Further, various message flow operations such as summarization and transforms may be specified to run within the infrastructure to create new derivative message flows. Unlike a tuplespace, there is no central store in a MOM system, and a MOM diagram looks more like a directed-acyclic graph (DAG). A MOM system differs from publish-subscribe in that message flows typically pass through several routing nodes across a wide-area network (WAN) and may be transformed as they do so.

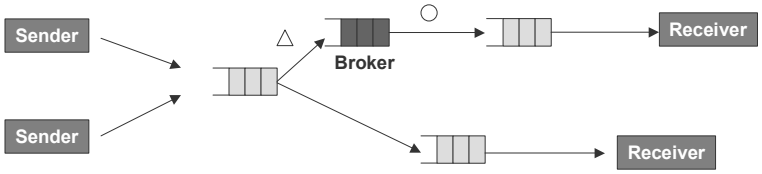


Figure 11 –Message Oriented Middleware System Diagram

Figure 11 shows conceptually how MOM systems work. Senders post messages into queues running on servers somewhere on the Internet. These messages are forwarded from server to server until they reach the receive queue for an appropriate receiving application. Brokers can run on servers in the network, receive one type of message, and then perform some transform on the message data and generate a different type of message.

Since MOM systems vary quite a bit, it is hard to specify how they compare on the coordination properties we specified. As a rule, they are not designed to be general-purpose coordination systems, and thus are not as expressive as is desirable for an interactive workspace. In addition,

since many are optimized for transactions and guaranteed delivery at Internet scales, latency is often sacrificed and the system may not provide perceptually instantaneity. Some of the systems are referentially coupled, requiring sender to specify receiver. Depending on the implementation, the API may not be very portable, and debugging may be difficult since transported data may be opaque (sometimes deliberately so through encryption of sensitive business information being transferred in the messages).

On the other hand, MOM systems do have some features that are desirable for interactive workspaces. MOM is designed to be extensible and provide for evolution over time, supporting snooping and interposition [27]. Applications are decoupled referentially and temporally, an important feature for any system tying together diverse systems across divisions within a company, or even multiple companies. MOM systems are designed to handle the interactions of thousands or tens of thousands of devices across the Internet so they can handle traffic loads that would be generated in an interactive workspace. Failure tolerance, guaranteed delivery, and transactions are some of the main features of MOM systems, so they meet the need for failure tolerance in an interactive workspace. The other two features, however, tend to cost MOM systems in latency. Finally, application portability is another key goal shared between MOM systems and what is needed by coordination infrastructures for interactive workspaces.

Two examples of MOM systems are IBM's MQ-Series [12] and Gryphon [3], which is an IBM research project.

Chapter 5 – The Event Heap Prototype Implementation

This chapter describes the prototype implementation of the Event Heap extended-tuplespace model for coordination in ubiquitous computing environments that was described in the previous chapter. The implementation bears the same name as the model, but in this chapter, 'Event Heap' is used to refer to the implementation. In places where this could lead to ambiguity, 'Event Heap model' will be used to refer to the model.

The Event Heap implements all the features described in Chapter 4, Section 4.2, and therefore has the properties described in Chapter 4, Section 4.1. In some cases implementation details directly affect the degree to which certain properties are satisfied. For example, failure tolerance (**P9**) and modular restartability (**F14**) are supported by implementing auto-reconnect for clients. Such cases are noted in the text.

Versions of the Event Heap have been available and deployed for three years (since Fall of 1999) and there have been three major versions. The initial version was a proof of concept and was built in C++ on top of T Spaces [145]. Since T Spaces is Java only, a JVM was instantiated for each client application, and calls were made into the T Spaces client using JNI. The server was an off the shelf T Spaces server—the only Event Heap code was in the client.

A second version, called Event Heap v1, was also built on top of T Spaces, but this time it was written in Java. A C++ version was again provided using an embedded JVM, but this time calls were made to the Java Event Heap implementation instead. By keeping the main body of code entirely in Java, version creep between the Java and C++ versions was avoided since all code modifications that didn't change the API only needed to be made on the Java side. As with the initial prototype, version 1 used an unmodified T Spaces server on the server side.

Both the initial prototype and version 1 exhibited very poor performance and lock-ups when more than about 100 events were present on the server. Since the problem could have been the Event Heap client code or T Spaces, and we didn't have access to the T Spaces source code, we were never able to isolate the problems.

Problems with the T Spaces server and lessons learned in the first two implementations, led to the development of the current version, Event Heap v2. Both the client and server are implemented from scratch in Java using only libraries that are part of the Java platform (JDK). A compatibility-shim in the form of a Java library is available that replaces the Event Heap v1 library and T Spaces libraries. This allows existing applications that use the C++ or Java Event Heap v1 API to work under the new, faster, and more reliable version. Throughout the chapter, the Event Heap v2 implementation is discussed with the exception of the C++ support, which currently is provided using the Event Heap v1 API and the compatibility-shim.

The material presented in this chapter is derived in large part from [84].

5.1 Event Description

The event is the basic unit of discourse with other applications using the same Event Heap server. Each event consists of a set of unordered, named fields. Since fields are unordered, they are always referenced by name instead of by index within the event. One special field with the name 'EventType' identifies the intended meaning of the event and a minimal set of additional fields that can be expected to be included with the event (this is to provide for the flexible typing feature, **F8**).

5.1.1 Field Structure

Each field contains the following:

- **Type:** The type for this field. Several platform independent types such int, float, and string are supported. Platform dependent types may also be used, but will only be accessible for the given platform.
- **Name:** The name of this field. It is intended to convey the meaning of the values contained in the field. This gives the implementation the self-description feature (**F7**).
- **Post Value:** The value for this field when it is posted to the Event Heap server. The post value is used for comparison when a template event is compared to a posted event.

- **Template Value:** The value for the field when the event is used as a template. This value is used to compare against the post values of events posted to the Event Heap when the event is sent to the server as a template.

5.1.2 Permitted Field Values

Values within each field may either be an actual value of the same type as the field (for example, "Hello, World!" if the field is of string type), or the value can be marked as one of several special types. The first of these is 'formal.' A 'formal' value matches any other actual or formal value when comparing events. The next special type is 'virtual' which indicates that the field should be ignored (treated as not existing) when this value is used in a match. Using 'virtual' allows developers to include extra optional fields when declaring Event Types without worrying about how they might complicate matching.

Two more special types are only used in conjunction with the routing fields discussed in Section 5.3 (examples of their use may also be found there). Neither of the values may be used by application developers for custom fields. The first type is 'auto-set.' Values with this type are automatically set by the Event Heap client code whenever the event is posted to the Event Heap server or used as a template to retrieve events from the server. If an application changes the auto-set value to something else, the Event Heap client will overwrite it when the event is used. The second special-type used for the routing fields is 'auto-set overrideable.' The only difference between this value type and 'auto-set' is that the user may apply their own value to the field and the Event Heap client will no longer automatically set it when the event is used. Table 8 provides a summary of the different value types that may be applied to the post and template values within a field.

Value Type	Meaning
Actual	This is a real value of the type for the field.
Formal	The value is a wildcard and matches any other value.
Virtual	Treat the field as not existing.
Auto-set	Value will be set by Event Heap Client.
Auto-set Overrideable	Value will be set by Event Heap Client if not set by the application.

Table 8 – Allowed Types for Values in Event Fields

5.1.3 Developer Flexibility

Allowing both post and template values, and allowing any given value to be 'virtual' gives the developer more flexibility in declaring and using events.

By separating out post and template values, developers can specify the default behavior of events. For example, a color field could be set to default to 'blue' when posted, but default to formal (match anything) when used as a template. By storing both values in the field, other developers can retrieve an event from the Event Heap by matching its post values, and then immediately use the event as a template to retrieve new events without having to worry about setting fields back to 'formal' that had specific values when posted. In practice, this feature has not been used very much, and the differentiation between post and template values is somewhat confusing to developers. Given this, the added flexibility may not be worth the steeper developer learning curve. We will evaluate whether to keep this functionality should future versions of the Event Heap be developed.

The 'virtual' value gives flexibility by letting developers define optional supplemental fields for an event without impacting matching semantics. Further, the Event Heap client strips fields that have virtual values for both post and template when events are placed into the Event Heap. This means that developers don't pay a performance penalty for defining extra fields that they don't use most of the time.

5.1.4 Event Matching

Events are returned to clients by the server only if they are determined to match the template event specified by the client. We will call the template event *TE* and the candidate matching event *CE*. Once *TE* and *CE* are selected, matching proceeds according to the following rules:

1. *CE* only matches *TE* if they have the same string value for the 'EventType' field.
2. All fields that have a 'Virtual' post value in *CE* are ignored/discarded.
3. All fields that have a 'Virtual' template value in *TE* are ignored/discarded.
4. All fields of *TE* must have a field in *CE* with the same name and type. Call these fields TE_n and CE_n.
5. The post value of each CE_n must match the template value of TE_n.
6. Post values match template values if:
    a. One or the other is 'Formal.'
    b. They are both 'Actual' and contain the same value according to equivalency semantics for that type.

TE matches *CE* if and only if all of the matching rules hold true. Note that the matching rules allow template events to match candidates that contain a super-set of their fields in order to satisfy the flexible typing feature (**F8**). A corollary to this is that *TE* matching *CE* does not imply that *CE* would match *TE* (if *CE* has a super-set of the fields of *TE*, then when *CE* is used as a template to match against *TE*, *TE* will not have all of the needed fields to be able to match). Thus, matching is not commutative.

'Formal' values may be used in both posted events and template events. When 'formal' values are used in a field of a posted event, that field will match any value requested by a receiver. Conversely, when 'formal' values are used in a field of a template event, that field will match any value posted by a sender.

5.1.5 Standard Fields

All events have certain standard fields, some of which are optional. The fields fall into three general categories: required user fields, routing fields, and internal use fields. They are briefly described here, and details can be found in the wire protocol appendix, which is found on the supplementary CD-ROM included with the dissertation.

The first category is user fields. One of these is the previously discussed 'EventType' field, and the other is the 'TimeToLive' field. The 'TimeToLive' field is used by the application to specify how many milliseconds the event should be kept on the Event Heap server before it is expired. This helps provide for the limited data persistence feature (**F4**).

The next set of fields are the routing fields, which control event receipt according to five different attributes: application instance, application name, device name, person, and group. The fields come in pairs, with a source version and a target version for each attribute (for example, 'SourceDevice' and 'TargetDevice'). All of them are automatically set for users by the Event Heap client library, but can be overridden. Application instance, application name, and device name are always present, while person and group are normally 'virtual,' but can be used if the application sets the person and group with the Event Heap client software. The detailed mechanism of how these fields are used is explained in Section 5.3 after the API calls for interacting with the Event Heap server have been described.

Finally, there are the internal use fields. These are used by the infrastructure to handle certain bookkeeping tasks and include 'SessionID,' 'SequenceNum,' and 'EventHeapVersion.' The first two are used for insuring at most once, FIFO ordering of events (feature **F13**), the details of which are explained in Section 5.4. 'EventHeapVersion' specifies the version of the event format

and is used by the server to determine which standard fields should be present. It is primarily intended to accommodate potential future changes to the set of standard fields.

5.2 Event Heap Client API

This section describes the API for the Event Heap client, which is used by application developers in writing their applications. While all of the major API calls are presented here, derivative versions of the major API calls and most client local calls are not discussed individually. Table 9 summarizes the main Event Heap client API calls, and provides the equivalent Linda-style tuplespace call where applicable:

API Call	Linda Equivalent	Description
Server Connection Calls		
new EventHeap(machine, port)	N/A	Connect to an Event Heap Server
Event Posting Calls		
void putEvent(Event)	void out(tuple)	Place an event onto the server.
Event Retrieval Calls		
Event[] getEvent(Event[])	tuple rdp(tuple)	Retrieve a copy of an event from server
Event[] waitForEvent(Event[])	tuple rd(tuple)	Blocking version of getEvent
Event[] removeEvent(Event[])	tuple inp(tuple)	Read and delete and event from the server
Event[] waitToRemoveEvent(Event[])	tuple in(tuple)	Blocking version of removeEvent
Event[] snoopEvents(Event[])	N/A	Retrieve copies of all matching events from the server, ignoring ordering
Event[] getAll(void)	N/A	Retrieve copies of all events on the server, ignoring ordering
Query Registration Calls		
Registration registerForEvents(Event [], Callback)	N/A	Register to receive copies of any future matching events placed on the server
Registration registerForAll(Callback)	N/A	Register to receive copies of all future events placed on the server
void Registration.deregister(void)	N/A	Cancel a previous registration
Other		
void deleteEvent(Event)	N/A	Delete an event from the server that was previously retrieved
void clear(void)	N/A	Delete all events currently on the server

Table 9 – The Event Heap Client API Calls

5.2.1 Connecting to an Event Heap Server

One Event Heap server is intended to be run for each interactive workspace. Clients connect to the server by instantiating an Event Heap object. During instantiation, the machine and port of the Event Heap server are specified. For the duration of the Event Heap object, all API calls are addressed to that server. By having one Event Heap per interactive workspace, the scope of the coordination infrastructure is made to match the users perception of the physical extent of the room. This is designed to account for the bounded environment characteristic of interactive workspaces (**H1**).

5.2.2 Posting Events

The 'putEvent' call places the event passed to it into the Event Heap. This is equivalent to the 'out' call in standard tuplespaces. The client sets all of the auto-set fields before sending the event to the Event Heap server. Any fields that have both post and template values set to 'virtual' are stripped from the event before transmission to minimize overhead in sending the event over the wire.

5.2.3 Event Retrieval

There are six API calls for retrieving events from the Event Heap server. Four of the calls, 'getEvent,' 'waitForEvent,' 'removeEvent,' and 'waitToRemoveEvent,' return a single matching event. Each of these takes an array of template events and returns the oldest unseen event on the server that matches one or more of the templates. They also return the set of templates that successfully matched to aid the client in dispatching the returned event.

The 'getEvent' and 'waitForEvent' calls return a copy of the event, leaving the original on the server for other applications to retrieve, with the latter blocking until a matching event becomes available if no matching event is on the server at the time of the call. They are equivalent to the non-blocking and blocking versions, respectively, of the tuplespace 'read' operator.

The 'removeEvent' and 'waitToRemoveEvent' calls are the destructive versions of the previous two calls, with the latter being the blocking version. With these calls, the original event is deleted from the server before being returned to the client. These calls are equivalent to the non-blocking and blocking versions, respectively, of the tuplespace 'in' operator.

The remaining two calls, 'snoopEvents' and 'getAll,' return collections of events. Unlike the previous four calls, both ignore ordering, so it is possible that previously seen events will be in the returned sets. The 'snoopEvents' call takes an array of template events and returns a copy of

all events on the server at the time of the call that match one or more of the template events. It is most useful for detecting current state, for example if beacons are being used. The 'getAll' call is similar to 'snoopEvents,' except that a copy of all events on the server is returned. It is useful for getting a snapshot of the state of the Event Heap. The set of matching templates is not returned for either of these calls because there would be no clear mapping between the set of matching templates and the set of returned events.

5.2.4 Query Registration

The three query registration methods, 'registerForEvents,' 'registerForAll,' and 'deregister' allow applications to register interest in events and receive a callback with each matching event that is placed onto the Event Heap. This gives the implementation the query persistence/registration feature (**F5**).

To register for some events, the 'registerForEvents' call is used. An array of template events and a callback function are passed to the method. Anytime new events matching one or more of the templates are placed onto the server, a copy of the event is returned to the client Event Heap code. That code then calls the registered function, passing it the matching event along with the templates that were valid matches. When registration is used, only events placed after the registration are returned to the application—even if matching events are on the Event Heap server at the time of registration.

The 'registerForAll' call is similar to 'registerForEvents,' except that no template events are passed to it, and all events placed onto the Event Heap server will be returned. This call is useful for logging or monitoring all events being sent through the Event Heap server.

Both the 'registerForEvents' and 'registerForAll' calls return a handle to a registration object. The object has a single call, 'deregister,' which cancels the registration. After 'deregister' is called no further callbacks will be received for that registration.

5.2.5 Other Server-side Calls

There are two additional calls that allow interaction with the server, 'deleteEvent' and 'clear.' The 'deleteEvent' call is passed an event that was previously retrieved from the server and causes it to be removed from the server (if it is still present). The 'clear' call removes all events currently on the server, and is intended primarily for administrators. In the future, we plan to protect this call so that only applications with sufficient privileges may call it.

5.2.6 Miscellaneous Local Calls

In addition to the previous mentioned calls, there are a few client API calls that only effect the state of the local Event Heap client.

The first set of calls are accessors for the variables the Event Heap client uses to set the routing fields for both post and template events (see Sections 5.1.5 and 5.3). They allow you to get the current value of:

- Instance name
- Application name
- Device name
- Group name
- Person name

In addition, the value of group name and person name can be set any time by the application. Instance, application and device are either set automatically or explicitly when the application first instantiates an Event Heap object and connects to the server and cannot be changed after that.

Another three of these calls control a local debug printing system. One of these sets the current debug level, which controls the detail level of the debug information, being printed out. Another, 'debugPrintln,' is a replacement for 'printf' or 'println' that can be used in Event Heap applications to print out debugging statements. It is passed a severity level so that, depending on the current level of detail being printed out, the statement will either be printed or suppressed. A final call allows the output stream for all debug statements to be redirected from the normal standard error. This can be used to direct statements into a debug window, for example. In future Event Heap versions we hope to provide a mechanism for room administrators to remotely re-route the debug streams so that they can monitor all Event Heap enabled applications running in an interactive workspace.

There are also additional informational calls to retrieve the machine and port of the Event Heap server to which the application is connected, and the version of the Event Heap client API. A final call allows a timeout for the blocking server calls to be specified.

5.3 Auto-set Fields for Routing

As mentioned in Chapter 4, Section 4.2.3, one of the desired additional capabilities for a tuplespace-based system in an Interactive Workspaces is some means of routing tuples (feature **F3**). The Event Heap accomplishes this by providing the standard source and target fields described in Section 5.1.5, and then having the client code automatically set them when an event is posted or used as a template.

This works as follows: when events are posted, the source version of each field is automatically set by the Event Heap client code. For example, the 'SourceApplication' field is set to the name of the application. Receivers can then match on certain application names in that field to receive events from that application.

When an application retrieves using a template, the target version of each field is also automatically set (these fields have the 'auto-set overrideable' value discussed in Section 5.1.2, so clients may choose to set the values such that they receive events targeted at others). The 'TargetPerson' field, for example, might get set to 'Bob.' Sources can then set the 'TargetPerson' field to 'Bob' if they want their event to only be picked up by applications currently being used by 'Bob.'

If a source does not set the target fields, they default to formal, which means the event will be picked up by all receivers that match the rest of the fields correctly. Similarly, if a receiver does not set the source fields, they also default to formal, and the receiver will pick up all events that match the rest of the template field values that were specified.

To make the automatic routing fields function properly, the Event Heap implementation uses the 'auto-set' and 'auto-set overrideable' field values discussed in Section 5.1.2. The post-values for the source versions of the fields (e.g. 'SourceDevice', 'SourceApplication,' etc.) all have the 'auto-set' value and cannot be overridden. This insures that developers may not spoof their identity by overwriting the values in these fields. The target fields (e.g. 'TargetDevice,' 'TargetApplication,' etc.) and the template values of the source fields are all set to 'auto-set overrideable' so that the auto-set feature will be disabled when an application explicitly sets the field values. This allows sending applications to set the target fields to route their events to specific targets, and receiving applications to select specific senders from which to receive events. Further, receiving applications can override their target fields to receive events intended for others, which provides one of the mechanisms required for creating intermediary applications.

Overall, the automatic-routing mechanism and the associated fields allow flexible and standardized point-to-point, multicast and broadcast communication for applications using the Event Heap.

5.4 Implementation Details

As mentioned at the beginning of this chapter, the Event Heap is implemented as a client-server system. The server and reference client are both written using the Java 2 platform. There are also clients for other languages and other paths to access the events on the Event Heap server machine. This section gives details of the Java client and server implementations and briefly discusses the other access mechanisms.

5.4.1 Event Heap Wire Protocol

As stated in Chapter 3, Section 3.3.1, one of the main characteristics of interactive workspaces is heterogeneity of hardware and software (**T1** and **T2**). In order to provide support for that heterogeneity, we would like applications to be able to access the Event Heap from as many different platforms as possible. Of course, some restrictions must be made—the device must be able to communicate with other devices in the workspace. We chose that the minimal functionality needed for a device or platform to have Event Heap functionality was support for TCP/IP communication using sockets. In part, this decision stems from our frustration early on with using T Spaces [145], which had a proprietary socket-based wire protocol that limited us to only using their Java client.

The current Event Heap implementation therefore has a well-defined TCP/IP socket-level wire protocol, which defines all interactions with the Event Heap server. Currently the wire-protocol has only been fully implemented in the Java reference client, but a partial client has been completed in C++, which we hope to finish soon, and another student on our project, Jeff Raymakers, has independently completed a working send-only C++ implementation.

There are two main parts of the wire protocol. First, there is a specification of how commands, event collections and sequencing information can be sent over a socket connection in a package called a WireBundle. Second, there is list of WireBundle commands that can be sent to the server along with what information needs to be packaged into the bundle with each command. There is also a specification of what can be expected to be returned by the server for each command. Essentially there is one WireBundle command for every client API call discussed in Section 5.2.

Beyond basic send-response semantics, the Event Heap wire protocol also contains specifications of how the initial contact with the server is made, and provision for future protocol versions. One notable feature of the wire protocol is that it is fully asynchronous. Commands sent via wire bundles to the server do not, in general, receive an immediate reply, and multiple sent commands may receive responses out of order. Details of how this works in practice will be made clear in the next two sections on the client and server implementations. The full wire-protocol is described in the wire protocol appendix found on the supplementary CD-ROM, which is included with the dissertation., and is distributed with the Open Source release of the Event Heap and the rest of the iROS system (http://iros.sourceforge.net).

5.4.2 Client Design and Functionality

The Java Event Heap client sends and receives events on behalf of the client application. It also handles setting the auto-set fields in posted events, or events used as templates. An overview of the Event Heap client is shown in Figure 12.

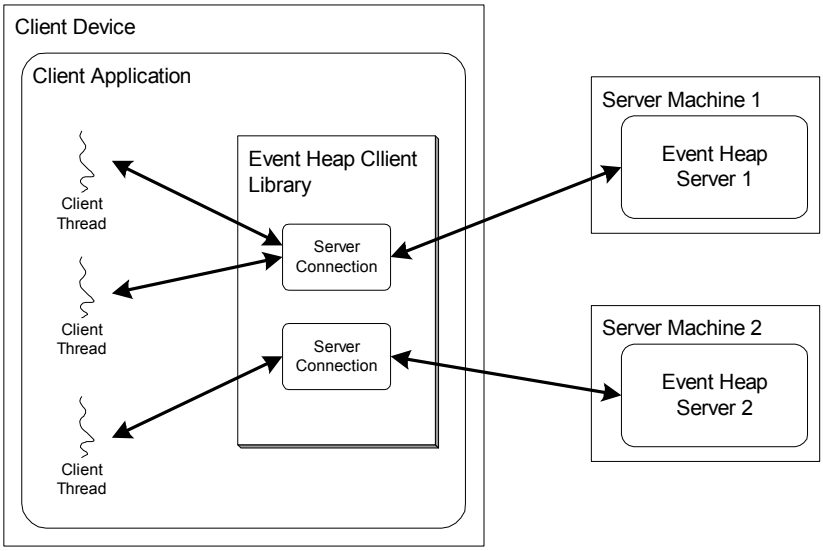


Figure 12 – Event Heap Client Implementation Overview

The basic operation is as follows. Whenever a thread in the client application creates a new EventHeap object (thus requesting a connection to some specific Event Heap server), the list of current server connections is checked. If no connection exists to the server, a new server connection object is created and connected to the requested Event Heap server. If a connection already exists, the new object shares the connection with other EventHeap objects connected to the same server. When a thread in the client application makes an Event Heap client API call that needs to access the server, the server connection object serializes the events, creates the

appropriate WireBundle object and sends it to the server. When the returned event or events arrive it passes them back to the client thread. Figure 13 shows in more detail how each server connection object works.

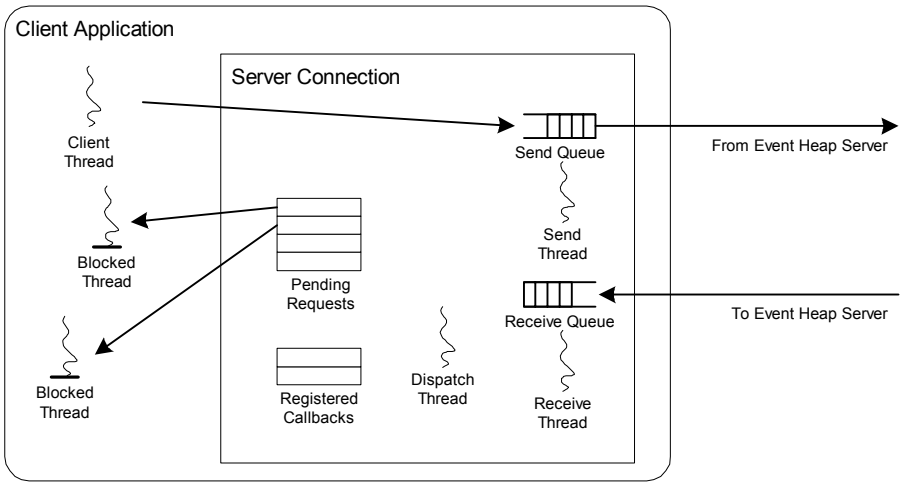


Figure 13 – Server Connection Object Implementation for Event Heap Client

As is shown in the figure, each server connection object has both a send and receive queue. Requests to the server from client threads are packaged into a WireBundle then placed into a send queue. A send thread forwards the WireBundles in the send queue out to the server over the socket connection as fast as possible. For all non-registration calls, the calling thread is blocked and a reference to the blocked thread is stored in a hash table using an ID for the request as a key. For registration calls, the callback is stored in a table of registered callbacks hashed by an ID for that registration, and control is returned to the calling thread.

On the receiving side, a receive thread reads returned WireBundles off the socket connection and places them into a receive queue. A dispatch thread processes WireBundles from the front of the queue and determines based on the return ID whether or not the returned result is for a registration. If it is for a registration, the appropriate callback is looked up in the table and the callback is invoked with the events in the WireBundle. For all other cases, the calling thread is looked up in the blocked threads table, and control is returned to the thread. For requests returning events, the events are returned. In the case of 'putEvent,' 'deleteEvent,' and 'clear,' only an ACK is returned from the server, so control flow is returned to the calling thread with no results.

In all cases, whether the return is coming via registration or a normal polling call, if only one matching event is being returned, the server returns that event alone and does not return the set of templates that successfully matched. This saves the bandwidth of returning template events that are already available on the client, and saves the server the processing cycles that would be needed to determine if more than one template was a match. Instead, the client compares the returned event with the templates passed in the original call and returns the set of matching templates to the blocked thread or as arguments to the callback.

One other important feature of the client is that it automatically reconnects. If the server is found to be down when a client thread makes a call, the client code continually tries to reconnect to the server, and completes the request when it succeeds. The current implementation waits randomly between zero and two seconds between retry requests to avoid taking too many CPU cycles. A client API call lets developers specify a timeout so they can regain control if the server stays down for too long. This part of the implementation gives it the modular restartability feature (**F14**).

5.4.3 Server Design and Functionality

The Event Heap server is a stand-alone Java application. It accepts connections from client applications, and then processes client requests and returns the appropriate results. Figure 14 shows the server architecture.

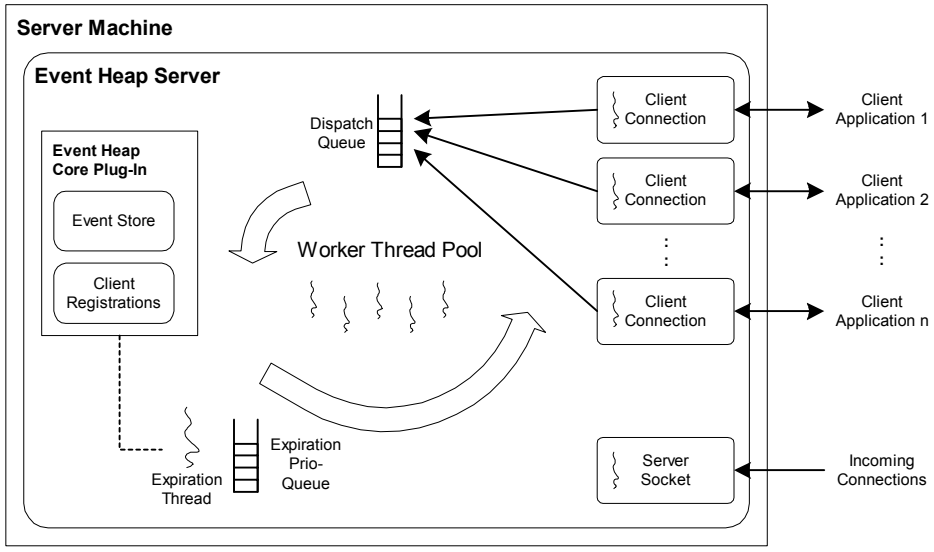


Figure 14 – Event Heap Server Implementation

As can be seen in the figure, the server has seven major components: a server socket, a set of client connections, a dispatch queue, a worker thread pool, the Event Heap core plug-in, an expiration thread and an expiration priority-queue.

The server socket is responsible for accepting incoming connections from new client applications. For each request, it creates a new client connection object to handle further communication with that client application.

Each client connection has a thread that reads WireBundles containing requests from the client application, and then places them into the dispatch queue for processing. For 'putEvent,' 'deleteEvent,' and 'clear,' all of which expect no returned value, an ACK is sent back to the client immediately after the request is placed in the dispatch queue. The client connections are internally quite similar to the server connection object discussed in the previous section, and, in fact, they share most of the same code.

The worker thread pool handles the processing of all requests from clients. Each thread repeatedly reads a request off of the dispatch queue, calls into the Event Heap core plug-in to process the request, and then returns the result, if any, to the client by making a call to the client connection object. The number of threads in the worker pool is specifiable via a parameter passed to the Event Heap server at startup.

The Event Heap core plug-in is where most of the real work of the server is performed. It is defined as a standard interface, which implements a method for each of the requests that can be sent from client applications. Any class that implements the Event Heap core interface may be passed to the server at startup as a parameter. Currently there are three implementations of the Event Heap core: a null version that just discards the passed in parameters and returns nothing, a test core that just stores all events in arrays hashed by EventType and uses brute force search, and a normal core which tracks previously seen or skipped events for each client and skips testing them for a match in future requests.

Each core (except the null core) is divided into two basic components: the event store and the list of client registrations. The event store contains the actual copies of events for the server, and the list of client registrations contains the sets of templates for clients that are either performing a blocking retrieve, or are permanently registered to retrieve copies of events. Whenever a retrieve request is received, the core first checks for a match in the event store, then, if no match is found and the call is blocking, it adds the request to the list of client registrations. If a registration request comes in, it is immediately added to the list of registrations.

When a 'putEvent' request comes in, the event being placed is first tested against each registration. If the event matches a blocked 'waitForEvent' call, it returns a copy to the appropriate client and removes the registration. If the event matches a blocked 'waitToRemoveEvent,' it returns a copy to the appropriate client, removes the registration, and marks the event as being deleted. If the event matches a permanent registration, a copy is returned to the appropriate client, but the registration persists. After all registrations have been tested, and if the event has not been marked as being deleted, it is then placed into the event store. Note that since all registrations are always tested, it is not possible for a permanent registration to miss any event posted to the server.

The final part of the server architecture is the expiration thread and expiration queue. Before the worker threads process the event through the event heap core plug-in, they add a deletion callback to the expiration priority queue with the time when the event should be expired. The expiration thread continuously sleeps until the time of the next expiration, then wakes up, pops the deletion callback off the priority queue and executes it, thereby deleting the event from the event store. The thread then sleeps until the execution time of the new head of the expiration priority-queue. To insure accurate deletion of expired events, the worker threads also check the deletion time of the head of the queue each time they insert a new deletion callback. If this time is later than the time for the deletion they are adding, they wake up the expiration thread so that it can calculate the new amount of time it needs to sleep and execute the just added deletion callback at the appropriate time.

5.4.4 Event Sequencing

To perform sequencing, each source tags every generated event with a Source, a SessionID, and a SequenceNum (sequence number). The Source needs to be unique among all sources participating in the Event Heap, so a random 32-bit integer is always appended to the source name (or instance name), which is specified by the calling application when the server connection is created.[9] The SessionID is also a 32-bit integer and is chosen randomly every time a source is instantiated. It 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. While using the random integers does not guarantee uniqueness for either Source or SessionID, the probability of collision is sufficiently low that they can be considered virtually unique.

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

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

5.4.5 Code Statistics for Event Heap Java Implementation

One desired property for coordination infrastructure for ubiquitous computing rooms was that the API be simple and portable (**P5**). The Java implementation shows that a full client can be implemented in relatively few lines of code and a small resulting JAR library. The Event Heap client code is 7,588 commented lines, or 1,158 semi-colons long. The distributed JAR file with only the client code is 48KB, which will easily fit on today's PDAs, and would fit on most current embedded devices (assuming the device had the appropriate JVM or native support for Java). Table 10 gives the code statistics for the prototype implementation in more detail.

Lines of Code	
Event Heap Server and Client	11,286 (1,909 semi-colons)
Event Heap Server	9,504 (1,632 semi-colons)
Event Heap Client	7,588 (1,158 semi-colons)
JAR Library Sizes	
Full iROS including non-Event Heap components	300 KB
Event Heap Client, Server and Debugger	109 KB
Event Heap Client Only	48 KB

Table 10 – Code Statistics for Event Heap Java Implementation

The LIME project reports that their extended tuplespace system implementation [120] is similar in lines of code and Jar file size, suggesting that our implementation is a reasonable one. Specifically, their Jar files are 120 KB, and they have 5,000 lines of non-commented source code.

5.5 Supported Hardware and Software Platforms

A key design goal is supporting a heterogeneous collection of machines and legacy applications (characteristics **T1** and **T2**). To do so, we have implemented a variety of ways for applications to access the Event Heap as shown in Figure 15. As discussed in detail earlier, the reference client and server implementations of the Event Heap are in Java. There is also a C++ Event Heap client under Windows, which works by instantiating a JVM inside the C++ application and making calls to the Java reference implementation using JNI. A full native C++ implementation is underway. A member of the Interactive Workspaces infrastructure sub-group, Shankar Ponnekanti has written a Python client, which works by wrapping the Java reference client.

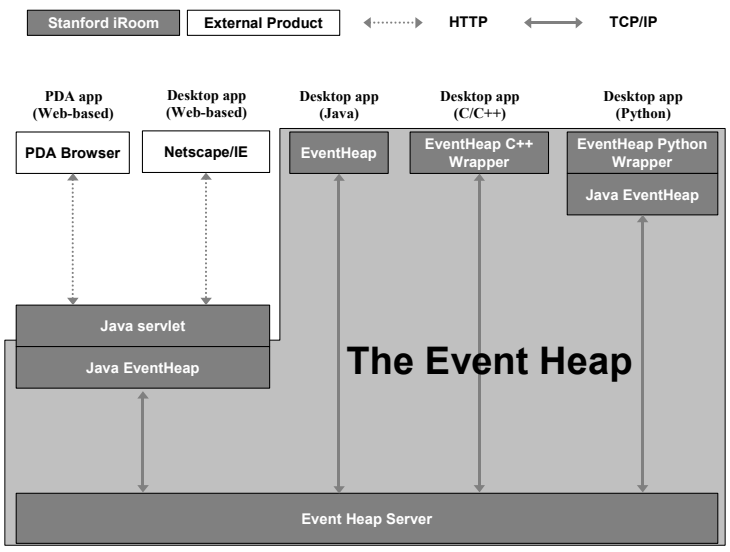


Figure 15 - Methods of Accessing the Event Heap

As other projects have discovered [94, 119], it is worth leveraging the web since web browsers and the web infrastructure are widely deployed. We therefore 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 (also written by Shankar Ponnekanti), and has allowed many basic interactions to be easily prototyped by simply creating a web page with the appropriate event submission URLs. A simple web form interface is available to assist users in constructing the specially formatted URLs for event submission. This path has proven useful in allowing PDAs to participate in controlling the iRoom, since even Palm-type devices now have web browsers available.

Using the currently available paths and software API's, the Event Heap is currently supported in some form on Windows, Macintosh OS X, Linux, Palm OS, and Windows CE. As mentioned earlier, there is also a well-defined TCP/IP protocol for communicating with the Event Heap server, so it is relatively straightforward to write a client for any new platform that has socket support.

5.6 Event Heap Debugger

Debugging the interactions of applications that use the Event Heap is theoretically simplified since all coordination is through a logically and physically centralized system. In practice, however, tools are required to allow monitoring of the communications. Therefore, to aid in debugging, we created the Event Heap debugger, shown below in Figure 16. Having the debugger helps provide for the easy debugging property (**P6**).

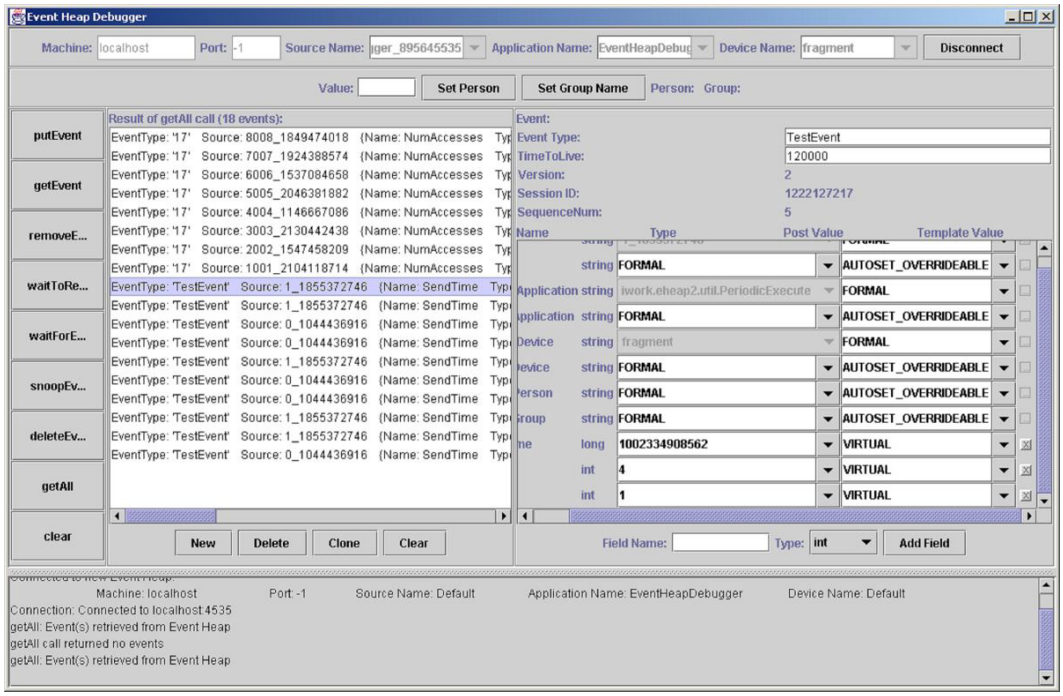


Figure 16 – The Event Heap Debugger

The debugger application is essentially a GUI interface to the entire Event Heap client API, with the exception of registration. It is implemented in Java using the Swing UI toolkit, and can be run as a stand-alone application or as a Java applet. The latter is particularly useful when the Event Heap server machine also runs a web server, as an unsigned version can be served which can interact with that Event Heap instance.

Monitoring the Event Heap with the debugger is straightforward. Fields can be filled out indicating the Event Heap to which to connect, and a connection is created by clicking on a button. Once connected, events can be constructed via a simple user interface. These events can be posted by clicking on the 'putEvent' button, or used to retrieve events by clicking on one of the five event retrieval buttons. To monitor the current state of the Event Heap, the 'getAll' button can be used.

One technique that is commonly used by developers is to do a 'getAll' to observe events currently on the server and determine the type of event some application is using. The event can then be modified and re-posted to trigger related behavior. For example, one developer didn't know the format of events to trigger display of web pages on the SMART Boards in the iRoom, so he used the debugger to look at the events in the Event Heap after having submitted a web page display request from another application. He modified the event and re-posted it to verify it worked, and then modified his own application so that it was able to submit web-page display requests.

5.7 Example Code

This section contains actual code for two simple applications that use the Event Heap. One is an application that sends out an event with some text that is meant to be rendered by a speech synthesizer somewhere in the interactive workspace. The other picks up these speech request events, synthesizes the audio, and plays it back to the computer on the local machine using the Java speech API. The two applications are designed to show the simplicity of minimal Event Heap applications. Since they represent the minimum amount of code to send and receive events, they also show how little code would be needed to be added to a legacy application to enable it for a basic level of coordination in an interactive workspace.

5.7.1 Sender Application

The sender application is only 29 lines of code long, and only 5 of those are for interaction with the Event Heap. Essentially the application makes a connection to the Event Heap server, creates an event, sets two field values in the event, and then posts the event. The code is as follows:

```
import iwork.eheap2.*;

class SpeakText
{

  static void main(String []args)
  {
    try{

      // Connect to Event Heap
      EventHeap theHeap=new EventHeap(args[0]);

      // Create event and set fields
      Event myEvent=new Event("AudioEvent");

      myEvent.setPostValue("AudioCommand", "Read");
      myEvent.setPostValue("Text", args[1]);

      // Put the event into the Event Heap
      theHeap.putEvent(myEvent);

    }
    catch(Exception e) {
      e.printStackTrace();
    }

  }  // end of main

} // end of SpeakText
```

5.7.2 Receiver Application

The receiver application is only slightly more complicated than the sender. It is 61 lines of code long, but again only five of them are Event Heap related. This time the application creates a connection to the Event Heap server, creates a template event, sets one field in the event, and then loops forever waiting for the speech event, extracting the text to speak and calling the routine to synthesize and play back the text. The code is as follows:

```
import iwork.eheap2.*;
import javax.speech.*;
import javax.speech.synthesis.*;

class Speaker
{

  static void main(String []args)
  {
    try{

      // Connect to the Event Heap
      EventHeap theHeap=new EventHeap(args[0]);

      // Create the template event
      Event myEvent=new Event("AudioEvent");

      myEvent.setTemplateValue("AudioCommand", "Read");

      // Loop forever retrieving events
      while(true) {

      try{

        // Block until a matching event arrives
        Event retEvent=theHeap.waitForEvent(myEvent);

        // Get the text to speak out of the event and say it
        simpleSpeak((String)(retEvent.getPostValue("Text")));

      }
      catch(Exception e) {
        e.printStackTrace();
      }

      } // end of loop

    }
    catch(Exception e) {
      e.printStackTrace();
    }

  }  // end of main

  private static void simpleSpeak(String phrase) {
```

```
    SynthesizerModeDesc mode = new SynthesizerModeDesc();
    Synthesizer synth = Central.createSynthesizer(mode);

    // Get ready to speak
    synth.allocate();
    synth.resume();

    // Speak
    synth.speakPlainText(phrase, null);

    // Wait till speaking is done
    synth.waitEngineState(Synthesizer.QUEUE_EMPTY);

  } // end of simpleSpeak

} // end of Speaker
```

5.8 Software Availability

The Event Heap is part of the iROS system, which is available for download and installation via Windows 2000 installers for both servers and clients (similar installers for Macintosh OS X have been completed and will be released shortly). The server install sets up a machine to run the Event Heap and associated web submission servlets, along with other iROS components. Each client is installed with pointer/keyboard redirection software called PointRight [87], and a system for moving web content between machines in an interactive workspace called Multibrowsing [88] (both applications are discussed in more detail in Chapter 6). The software install has been designed to be relatively painless, and a recent two-laptop system install was completed to the point where the system was running in less than fifteen minutes.

Once the software is installed, the PointRight system allows any iROS client machine to control the keyboard and mouse input to machines displayed in the environment by treating them as a large virtual desktop. The multibrowsing tools allow web content to be easily moved among machines in the workspace. With the iROS framework running it is also straightforward to begin writing new applications or modifying older applications to allow them to coordinate in an interactive workspace.

To allow others to use and extend it for their own locations, the Event Heap code along with the code for the rest of the iROS system and the distributed applications is available as Open Source. The code can be found on the supplementary CD-ROM included with the dissertation., or is also available through http://iros.sourceforge.net.

Chapter 6 – Evaluation

There is no straightforward way of evaluating software infrastructure for interactive workspaces since there exist no clear metrics. Such infrastructure is designed primarily to facilitate the tasks undertaken by the users of interactive workspaces by providing an appropriate set of tools for developers. As such, unlike in other system areas where processor speed or latency can be used as the primary metric, there is no clear property or set of properties that can be quantitatively measured. Further, since the field itself is still developing, there is no large base of other infrastructures against which a system can be tested and compared. As Hull found in trying to evaluate a context management system [80], it is hard to measure the utility of these systems to the end user, since it is dependent on the applications that might be constructed, and social issues, such as privacy, that could affect perceived desirability.

Despite the lack of clear-cut methods of evaluation, there are nonetheless several ways of evaluating the Event Heap model, both as a general set of properties and features for coordination in interactive workspaces, and as a specific implementation. This chapter presents an evaluation of the following three main contributions of this dissertation:

- **C1:** A characterization of interactive workspaces with regards to a coordination infrastructure.
- **C2:** The set of system properties and design features that comprise the Event Heap coordination model for interactive workspaces.
- **C3:** An implementation of the Event Heap model for proof of feasibility.

Three different means of evaluation are used in the remainder of the chapter to validate the contributions, and point out places where we made incorrect assumptions. The following types of evaluation are used:

- **Evaluation through usage:** The Event Heap, along with the iROS system of which it is a piece, has been deployed in interactive workspaces all over the world. The Event Heap itself has also been used in the construction of dozens of applications that are run in one or more of these interactive workspaces. We believe that the Event Heap is better suited for coordination in interactive workspaces than any existing system. While documenting the successful use of the system cannot prove this to be the case, it does provide supporting evidence, since a poor coordination system would be quickly abandoned by developers and those deploying the system. We also make observations about the deployments and application implementations with regard to the proposed system properties and features and how useful they have been in practice.

- **Evaluation through user survey:** We gave a survey to developers who have used the Event Heap and to administrators of iROS based interactive workspaces. The survey asked for feedback on the characteristics of interactive workspaces, system properties for coordination infrastructure they found important, and experience with the Event Heap implementation. The results of this survey indicate that for the most part the characteristics we posit for interactive workspaces and the model of coordination we propose fit well with the needs and beliefs of actual developers and administrators. Further, developers and administrators were satisfied with the actual Event Heap implementation and believed that it had simplified their work with interactive workspaces. In some cases, developers stated that using the Event Heap implementation had enabled them to pursue projects that would have otherwise not been feasible.

- **Evaluation through performance measurement:** While performance metrics alone are insufficient to evaluate the utility of coordination infrastructure in interactive workspaces, there are nonetheless certain minimal levels of performance needed to give the end users a satisfactory experience. Performance measurements were therefore made of the prototype implementation that show it to more than meet the requirements for coordination in an interactive workspace.

The remainder of this chapter evaluates contributions **C1** through **C3**, according to the three techniques of evaluation just mentioned. One section is dedicated to each means of evaluation.

Application or System Name	Description
Systems	
Data Heap	Data storage and transformation system for interactive workspaces [92].
iCrafter	Interface generation system for interactive workspaces [122, 123].
iStuff	Collection of wireless input and output devices that can be mapped as input to Event-Heap-enabled applications [33].
PatchPanel	Provides transformation from one event format to another. Can be dynamically programmed using events.
Applications	
The CIFE Suite	A collection of applications for construction management [101, 102].
Multibrowsing	Allows movement and coordination of web content among displays in an interactive workspace [88].
SmartPresenter	Allows coordinated presentation of PowerPoint slides and other material across the displays in an interactive workspace.
PointRight	A flexible pointer and keyboard redirection system for interactive workspaces [87].
WorkspaceNavigator	Records activity and content during a collaboration session in an interactive workspace. Allows browsing and usage of captured content in future sessions. [82]
iClub	Collection of applications that turn the iRoom into an interactive dance club.
Red Board	Allows users to 'login' to the iRoom by swiping a bar code. Once logged in, the system opens a data and settings folder for the user.
Post-Brainstorm	Brainstorming and white-board tool for large, high-resolution displays. Includes a camera-based scanner for capturing paper-based content onto the high-resolution display.
GroupBoard	Allows for collaborative sketching and brainstorming with multiple-users and PCs.
iWall	A custom, multi-user display environment that will run across one or more displays in an interactive workspace. Currently in development.
Virtual Auditorium	A video conferencing system for education that allows for one teacher to connect to many students. Student video feeds are displayed across the displays in an interactive workspace and can be dynamically moved among the displays [41].
iPong	The classic 'Pong' game, but it can be played across one or more displays in an interactive workspace, and paddle control can be flexibly remapped through the Event Heap.
Image Manager	Application component that can send and receive images and then allows for their manipulation using gestures.
Radiology Visualization	Integrated viewing and manipulation of 2D and 3D radiology data across multiple displays in an interactive workspace.
Storyboarding	A system for quickly moving from storyboard sketches to a roughed up storyboard movie. Integrates scanners, image sorting and manipulation and Adobe Premiere across multiple machines in an interactive workspace.

Table 11 – Applications Written Using the Event Heap

6.1 Usage and Deployment

Our first method of evaluation of the Event Heap model and the actual implementation is to examine whether and in which applications it has been used, and in what sorts of environments it has been deployed. In another situation where there were no obvious or straightforward evaluation mechanisms, Coen suggested that the following question could be applied to the evaluation of the MIT Intelligent Room and its software [46]: "Do people bother to use it?" If a system is more of a hindrance than a help, people will continue to do things the way they have always been done. The goal of this section is to show that people do "bother to use" the Event Heap for writing applications and they have deployed it in various environments, suggesting that as a coordination system it is more of a help than a hindrance.

6.1.1 Applications and Systems Using the Event Heap

The Event Heap has been used in the development of tens of applications and systems. Since it is now available as Open Source as part of the iROS system, it is being used by people all over the world. In particular we know that it is being used by our collaborators at KTH in Sweden, and that it has also been used in two locations in Germany: the Learning Lab of Lower Saxony (L3S) [9] and within the Ambiente Group of GMD-IPSI in Darmstadt. At this point, the Event Heap is being used too widely to compile a complete list of applications that use it, but we give a partial list in Table 11.

The remainder of this section describes some of these systems and applications in more detail—some of the applications are omitted for sake of brevity. For each application or system, a description is given followed by implementation details relevant to how the Event Heap is used. Finally, the key features of the Event Heap model that enable the application are discussed.

Systems Using the Event Heap

The following four systems all build on top of the Event Heap to provide additional functionality to application developers in interactive workspaces.

Data Heap

The Data Heap is the component of the iROS system that provides for data movement, one of the important capabilities needed by users of interactive workspaces discussed in Chapter 3, Section 3.4. It was written by Emre Kıcıman, one of the members of the infrastructure sub-group of the Interactive Workspaces project.

The Data Heap allows any application to place data into a store associated with the local interactive workspace. The data is stored with an arbitrary number of attributes that characterize it, and data can be retrieved by a query specifying attributes that must be matched. By using attributes instead of locations, applications need not know which specific physical file system is storing the data. Storing data by attribute has been previously used in the Presto system [55].

In addition to storage, the Data Heap provides a means to transform data. This functionality can either be used when no application is available to display the data in its native format (e.g. a PhotoShop document in an application that only supports JPEG images), or to reformat the data for a device with a different output format (e.g. using speech synthesis to output a text file through a speaker). Data is posted to the Data Heap including a format attribute, which specifies the type of the data. When requesting data, applications specify a set of formats that they understand. If the native format of the matching data file is not in the set of valid formats for the requesting device, the Data Heap will automatically transform it into a supported format before returning it to the application. This automatic translation makes data more portable among the diverse applications and devices in an interactive workspace.

The Data Heap frees data producers from having to know in advance the capabilities of consumers, and thus from having to know who the consumers of their data will be. This property is essential in a multi-platform ubicomp environment in which not all clients agree on a canonical set of data formats.

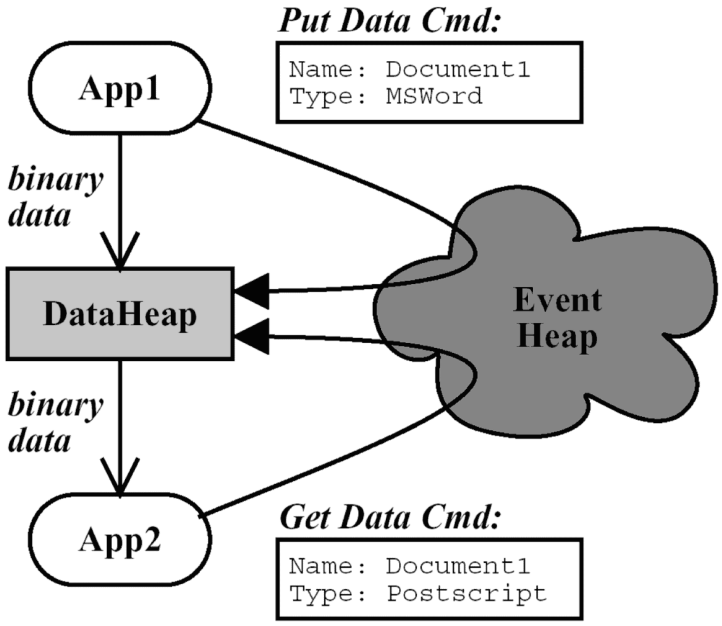


Figure 17 – The Basic Data Heap Mechanism

The basic mechanism used by the Data Heap is shown in Figure 17. The attribute meta-data is stored and queried using a fast, in-memory XML database. The actual data is stored on a Web-DAV [68] server. Using the Event Heap, data producers indicate their desire to store a document and its associated metadata, and consumers query for metadata and indicate which formats they can accept. The Data Heap responds to consumer queries by dynamically instantiating a chain of transformation operators to convert the data to one of the acceptable types (the transformation engine was published in [92]). Data transfer of the transformed result uses HTTP, the native protocol of the WebDAV server. HTTP is also used for initial placement of data into the system.

The primary coordination type used by the Data Heap is essentially a remote procedure call (RPC). The Event Heap doesn't provide any special functionality beyond other coordination models in this area since RPC is supported by most of them. One benefit of using the Event Heap is that applications are assured of using the correct Data Heap for an interactive workspace as long as they are accessing the Event Heap for that workspace. Further, debugging of the system is enhanced since developers or administrators can monitor Data Heap calls using the Event Heap debugger. The limited data persistence of the Event Heap also insures that transient failures of clients using the Data Heap, or of the Data Heap server itself, will be masked.

iCrafter

The iCrafter system [122] provides a system for service advertisement and invocation, along with a user interface generator for services. It was designed and implemented by Shankar Ponnekanti, a member of the infrastructure sub-group of the Interactive Workspaces Project. It is the component of the iROS system that helps provide for the *moving control* capability described in Chapter 3, Section 3.4.

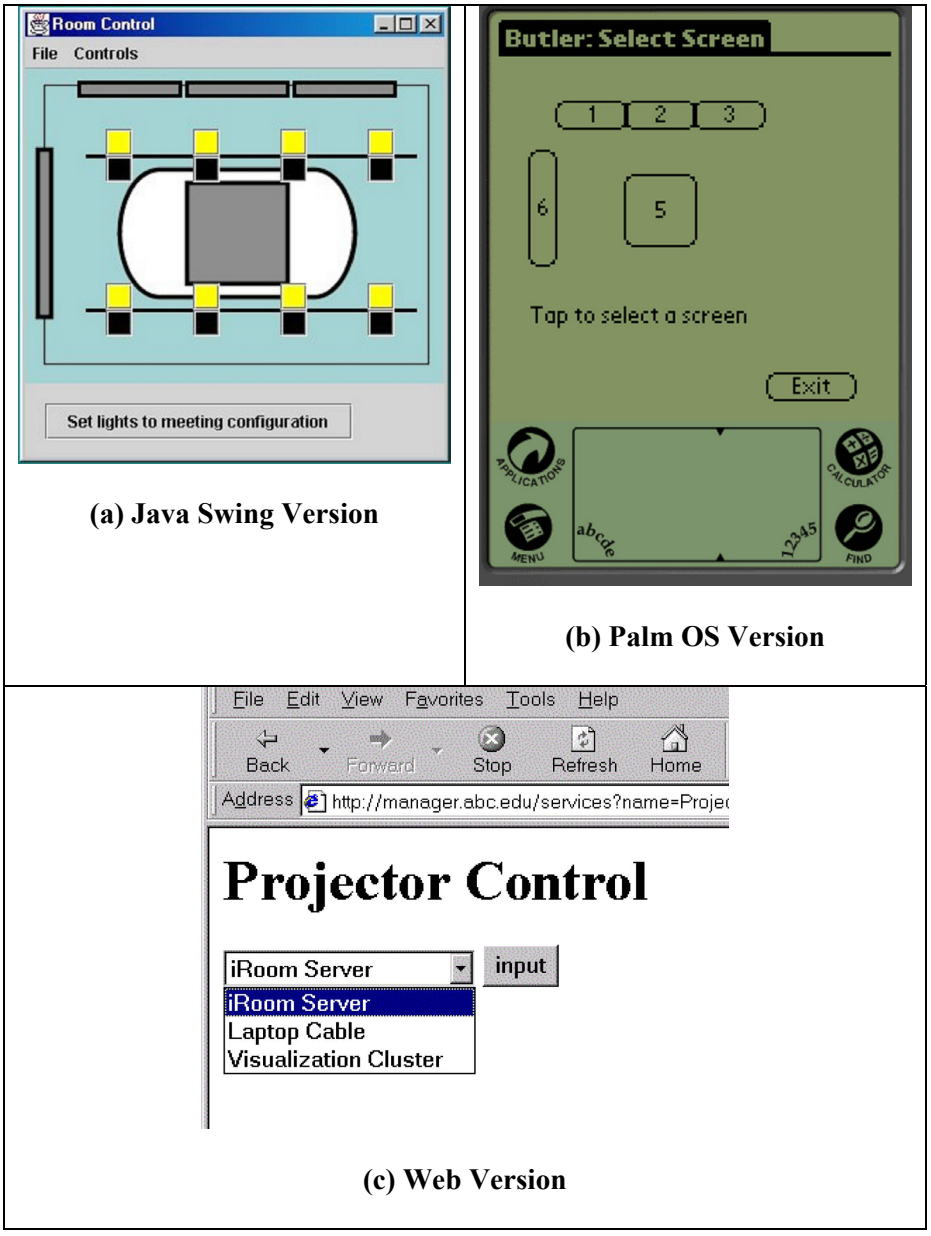
In the iCrafter system, services make announcements of their availability in the local interactive workspace. The announcements specify the type of the service, and also contain a service description, which specifies the invocable actions for that service.

iCrafter includes an interface manager service that provides a list of other services currently available to users of the interactive workspace. Users may request an interface for a specific service, or for a collection of services, to be run on any device they are using in the workspace. The interface manager will then generate and return to the user the most appropriate interface for the combination of services being controlled and the controlling device.

iCrafter supports a range of interfaces from those that are custom designed to those that are fully generic. When a custom-designed interface is available for a device/service pair, it will be sent.

Otherwise, a more generic generator will be used to render into the highest quality interface type supported on the device. Generated interfaces may also take advantage of context information for the local workspace.

For example, if room geometry is available, a light controller can show the actual positions of the lights in a graphical representation of the workspace.



(a) Java Swing Version

(b) Palm OS Version

(c) Web Version

Figure 18 - Various iCrafter Generated Room Controllers

One of the most commonly used iCrafter interfaces in the iRoom is the Room Controller. It provides a geometric view of the room, the ability to move data to any display in the room, and to

control lights, projectors, and routing laptop video to screens. Some of the different Room Controller interfaces generated by iCrafter are shown in Figure 18.

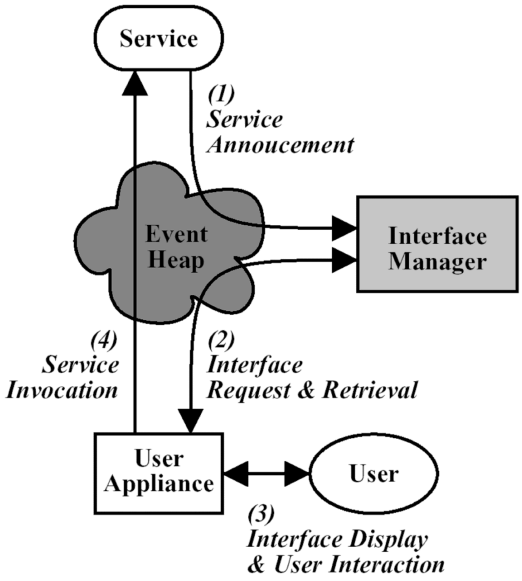


Figure 19 - Basic iCrafter Interaction

Figure 19 illustrates how the iCrafter framework works. Services periodically beacon their descriptions to the Event Heap with an expiration time set to two times the period (step 1 in Figure 19). This provides a mechanism for devices that are removed or crash to automatically be removed from the list of valid services. Service Descriptions describe the services' programmatic interfaces in an XML-based service description language. Unlike [63, 78], iCrafter service descriptions do not include URLs or physical addresses of the services or UIs, because this would require re-mapping whenever service or UI locations change.

When a request for an interface arrives from an appliance (step 2), the interface manager chooses a custom UI generator if one is available for the service and appliance platform, or a generic service-independent UI generator otherwise. The generators can access a local XML database for the local workspace context information (such as physical locations).

When a user performs an action on the user interface (step 3), the service is invoked by placing an event into the Event Heap (step 4).

The iCrafter architecture is entirely based on the Event Heap model. Compared to similar previous work [63, 78, 141, 145], this offers the ability to snoop on and interpose between calls to ICrafter services. Services are also temporally decoupled from their controlling interface which helps mask any transient failure.

The soft-state beaconing that is used by services through the Event Heap insures that services that crash will become unavailable as soon as their last beaconed event expires from the Event Heap. Unlike traditional beaconing systems, individual clients need not cache the most recent beacons but can check the Event Heap server instead.

As with the Data Heap, by using the Event Heap interface, clients are assured that they are communicating with the appropriate interface manager and are seeing the services for the local interactive workspace.

PatchPanel

The PatchPanel is a system designed by Tico Ballagas, one of the students working on the Interactive Workspaces Project. It is a generic event interposition agent that works over the Event Heap that was designed primarily to support the iStuff components [26] described in the next section.

One of the properties of the Event Heap model is extensibility (**P3**), which is provided in part through snooping and interposition (which is in turn provided by logical centralization, transparent communication and referential decoupling). Normally, a custom interposition agent would need to be created every time event transformation was needed to allow coordination between two applications with incompatible event formats.

The PatchPanel provides a standard mechanism for accomplishing this transformation by maintaining a table of active event transformations. It registers to receive events using templates listed in the table. When it receives an event, the PatchPanel applies the appropriate mapping from its table of transformations to create an event with the new type which it then sends out over the Event Heap. The easiest mechanism for setting up transformations is via a simple graphical user interface application written in Java.

One unique feature of the PatchPanel is its recursive nature. The table of transformations itself can be modified by submission of PatchPanel events over the Event Heap. Thus, some event can be setup to transform to a PatchPanel event, which changes the transformation setup for other types of events. For example, a user interface is set up with a slider that emits slider events and is run on some device in the workspace. The PatchPanel is set up so that slider events transform to volume adjustment events, allowing the slider to control the volume of audio in the workspace. A second UI, which can be brought up on another machine in the workspace, consists of a button, which emits button events. Button events are set up to map to PatchPanel events to reconfigure the transformation of slider events. Specifically, slider events are set to map to light-dimmer

events. Thus, when a user hits the button, the slider switches to work as a light dimmer instead of a volume control.

Obviously, the PatchPanel relies heavily on the Event Heap infrastructure. Without the extensibility property (**P3**) provided by the transparent communication, referential decoupling and logical centralization of the Event Heap the PatchPanel would not be possible. The PatchPanel, in turn, provides a simple mechanism for extending and inter-connecting arbitrary Event Heap based applications in an interactive workspace.

iStuff

iStuff [26, 33] is a toolbox of wireless, physical user interface components that provide paths into and out of the Event Heap. It is being developed by a group of HCI researchers in the Interactive Workspaces Project. Their primary goal is to "provide a prototyping toolkit for building experimental HCI research prototypes that can be assembled, reconfigured, and disassembled quickly, potentially even within a single experimental session, to test alternative ubicomp interface solutions."

Table 12 lists some of the types of components they are investigating:

	Input Devices	**Output Devices**
One-bit	Push-buttons, binary sensors like motion detectors	Status lights, buzzers
Multi-bit	Rotary switches, digital joysticks, barcode scanners, fingerprint readers	LED arrays, alpha-numeric LCDs
Near-continuous	Sliders, trackballs, sensors	Servos, motors, dimmable lights
Streaming	Microphones, cameras	Speakers, small-screens

Table 12 – Categories of iStuff

In developing iStuff they have several key goals:

- Completely autonomous packaging (wireless and battery powered).
- Seamless integration into existing ubicomp environments.
- Easy configuration of mapping between devices and functionality in the environment.
- Simple and affordable circuitry that can be easily built by other HCI teams.

iStuff can be combined together with applications or other iStuff in the iRoom using the previously discussed Patch Panel.

As one example, the iButton is a one-bit, push-button iStuff. When pressed, it sends a unique ID. The base station maps the ID to sending an event to the Event Heap. A simple application picks up that event and maps it to a script that can send a bunch of other events.

Using this system, they set-up a RoomStarter button in less than fifteen minutes. When the button is pressed, all the lights and projectors are turned on, a web introduction the room is loaded on one of the screens, and various group files are loaded on the other screens.

Another iStuff object is the iPen, which is an iButton shaped like a pen. The pen allows button input to standard touch screens. For example, the SMART Board touch screens in the iRoom have no clean mechanism for a right-click. The iPen has therefore been setup such that when users press the button before using the pen on the SMART Board, the following touch interaction with the pen is interpreted as a mouse motion with the right-button down (as opposed to normal touch-interaction which is interpreted as left-button down interaction).

The key enabler of iStuff is the ability to map all inputs and outputs of the wireless components through the Event Heap. Since most iStuff devices are simple transmitters or receivers, they cannot host an Event Heap client themselves. Each iStuff component therefore has a proxy that represents it on the Event Heap. In the case of an input device, whenever the proxy receives a transmission triggered by a user from a device, it posts an event with the iStuff ID of the component along with the appropriate values as an event on the Event Heap. Similarly, if an event is received by the proxy requesting that some output be triggered on a specific iStuff component, the proxy sends the appropriate radio transmission to that device to trigger the appropriate action.

To make the iStuff actually useful, the previously described PatchPanel is required. It maps specific iStuff input events to some other event format. For example, an iButton event can be mapped to trigger the opening of some specific URL by submitting a URL open event. Similar mappings can be made for iStuff output devices: an iSpeaker event to announce that the lights have been shut off can be created whenever a lights-off event is seen by the Patch Panel.

iStuff is almost entirely dependent on the extensibility property of the Event Heap model (**P3**). All meaningful interactions are created using the interposition features of the previously discussed PatchPanel. The referential decoupling provided by the Event Heap allows iStuff to be transparently used in place of or in addition to the original coordination mechanisms designed into applications. Since iStuff is designed as a toolkit to investigate HCI issues in the context of post-desktop user interfaces, it was also critical that the Event Heap have the perceptual instantaneity feature (**P7**). Without this feature, interactions with iStuff would be slow and frustrating.

Applications Using the Event Heap

The following are all applications or application suites that have been enabled by or constructed using the Event Heap.

The CIFE Suite

One early adopter of iRoom technology was the Center for Integrated Facilities Engineering (CIFE). They are a group of civil engineering researchers investigating how to automate the process of planning and managing large construction projects. Discussions with CIFE inspired the motivating scenario given at the beginning of Chapter 3. Our collaboration with them also led to the development by them of the CIFE application suite [101, 102], which enables the aforementioned scenario. The CIFE suite demonstrates how a collection of coordinating application components can be created that permit users to select which components are running at any given time. The CIFE Suite is also the best existing example of dynamic application coordination, one of the usage capabilities from Chapter 3, Section 3.4.

The CIFE researchers originally approached us looking for ways of moving beyond the paper based collaboration that occurs in traditional construction management and planning. Their observations revealed that "communicating project information through paper-based graphical representations limits the team's ability to work together to solve problems and make decisions." They further noted that "by their nature, project teams bring together participants from many disciplines that use discipline-specific information formats and discipline-specific modeling, analysis and visualization tools for their work." Many of these tools were legacy applications that would be difficult or impossible to port to some custom coordination framework. The similarity of their application needs to the goals of our project made implementing the CIFE Suite a good test of our coordination infrastructure.

CIFE designed a set of viewers for their data that could be run on the various displays in the workspace, and they have since built an Event-Heap-based interactive workspace of their own (see Section 6.1.2) where they also use the CIFE Suite. Some of the viewers they have created are:

- 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 view change events, object and zone selection events (e.g., building 3), and date change events.

- Another map viewer that highlights zones based on zone selection events.
- An applet based web viewer that displays tables of construction site information. This viewer also emits zone selection and date change events as table information is selected It also listens for the same events, and upon receipt, highlights the appropriate information in the table.

All of the applications are essentially standalone, but communicate through the Event Heap. The viewers all use common event types, such as date change, or change model view, so the various components of the suite retain their ability to coordinate while being displayed on any screen in a workspace. For example, when the current date is changed in the 4D viewer, it emits a date change event which is picked up by the applet based construction viewers, regardless of which machine is running them. Those viewers then display information for the date that was transmitted. The loosely coupled nature of the suite means that if no event source is running, or there is no event sink, it does not affect any of the application components currently in use.

The overall effect for the user is that clicking on an element in any one of the applications produces a corresponding change in the other viewers running in the workspace.

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 C++ code. The web applet viewer is a fully custom application. The construction site map that allows different views to be selected was created in Macromedia Flash [10] with embedded URLs that trigger view change events when a region of the map is selected. The Event Heaps HTTP event submission mechanism made it possible to create this final application with a minimum of development time.

The CIFE Suite leverages the referential decoupling and anonymous communication capabilities of the Event Heap to provide for a set of loosely interacting components. This allows users to run any sub-set of the applications in the suite anywhere in the interactive workspace and continue to have them interact with one another

Multibrowsing

The Multibrowsing system [88] was one of the earliest applications developed for and deployed in the iRoom. It provides a mechanism to move web content among the computers displaying in an interactive workspace, supporting the user's need to be able to move data as discussed in Chapter 3, Section 3.4. The author was one of the main architects of the multibrowsing system.

Multibrowsing provides three main mechanisms with which users can move web content around an interactive workspace:

- A web page or a link currently displayed in a browser on an enabled machine can be redirected to any other display in the environment without direct interaction with the target display.
- A web page currently displayed on any display in the environment can be “pulled onto" a local browser without the need for copying the URL and typing it in the local browser.
- Web pages can be purpose-designed for multi-display viewing by embedding links that open web pages on other displays in the environment.

The first two mechanisms work via options that are added to the right-click menu in Internet Explorer on the Windows operating system. The user can choose ‘Multibrowse to…” to select a currently active machine to which to push the loaded web page, or ‘Multibrowse from…’ to request a page being displayed on another active machine to load on the local machine. The ‘Multibrowse to…’ functionality is shown in Figure 20.

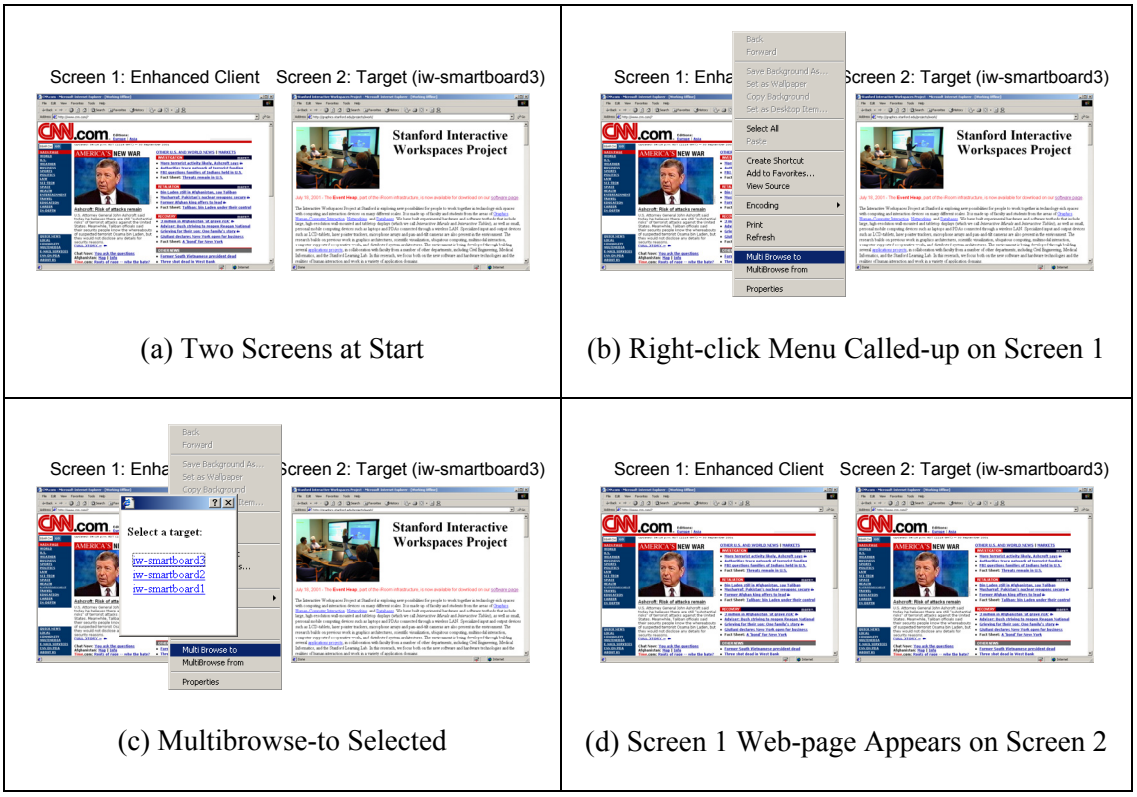


Figure 20 - Example of Multibrowsing

We have found these capabilities to be useful in the iRoom, especially for meetings and presentations. Multibrowsing has also proven useful in offices with multiple PCs.

In the multibrowsing framework, machines can assume one or more of the three different roles described below:

1. **Target:** A target runs a small daemon application (called the *butler service*) that listens on the Event Heap for multibrowsing events directing it to load a web page into a local browser window. The daemon also listens for requests for the URL of the web page currently loaded in the active browser window, and responds with a multibrowsing event containing that URL. Finally, the daemon sends out regular beacons over the Event Heap announcing that it can accept requests for pushing and pulling web pages.
2. **Regular Client:** A regular client need only have a web browser installed. It can access web pages that use the Event Heap *gateway servlet* HTTP event submission mechanism. By clicking links encoding such event submissions (called *MB fat-links*), it can trigger the display of a URL on the appropriate display.
3. **Enhanced Client:** An enhanced client has the appropriate plug-in (called *MB2Go*) installed within Internet Explorer to permit usage of the right-click menu to push and pull web pages around the interactive workspace. When users access the menu, it collects the beacons being broadcast by *targets* in order to display the list of available devices to the user.

The overall architecture of the multibrowsing system is shown in Figure 21.

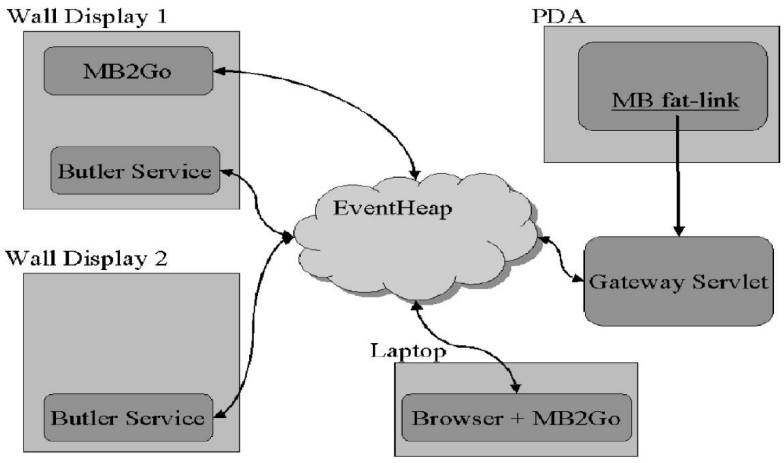


Figure 21 - Multibrowsing Architecture

The overall architecture of the multibrowsing system is shown in Figure 21. In the figure, wall display 1 acts as both a target and an enhanced client. Wall display 2 acts only as a target. The laptop acts as an enhanced client while the PDA acts as a regular client.

The multibrowsing system leverages the Event Heap in two key ways. Most obviously, it uses the Event Heap for the coordination messages that trigger the loading of web pages. While most of the time these requests are targeted at one machine, it is also possible to request that all displays in the space load a page by specifying the target display as formal (wildcard). This mechanism would be harder to support in a coordination system that only had remote procedure call RPC. The multibrowsing system also uses the Event Heap to beacon available clients. Since the scope of the Event Heap is limited to the local interactive workspace, clients are assured that all target machines are in the same workspace as themselves. Finally, the HTTP event submission mechanism of the Event Heap allows regular clients with only an off-the-shelf web browser to participate. Thus, handhelds, laptops, wall-mounted, and tabletop displays can all function as regular clients with no custom multibrowse software installed.

SmartPresenter

SmartPresenter is a multi-display, multi-object presentation program for interactive workspaces. It was designed and implemented by Emre Kıcıman and Penka Vassileva, members of the infrastructure sub-group of the Interactive Workspaces project.

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, in the iRoom a presentation might specify that at time-step 4, slide 17 from a Power Point presentation is shown on the left touch screen, a 3-d model is displayed on the high-resolution front screen, and web pages are displayed on the other two touch screens.

To set up the presentation, users create a script specifying which content to show on which displays for any point in the presentation. A web based configuration tool is available to assist users in creating the script.

While the presentation is running, a control UI can be run on one or more displays in the workspace. It allows users to step forward or backward in the running presentation. In addition, audience members may follow the presentation from a laptop by tracking the content of any particular display in the interactive workspace (provided they have the appropriate software to display the type of content being shown on the display that they are tracking).

PowerPoint and web pages are two of the default formats supported by the system. In addition to content, any arbitrary event may be sent over the Event Heap at a given point in the presentation, This makes it straightforward, for example, to trigger lighting changes or switch the video input of a projector during a presentation.

On the implementation side, SmartPresenter consists of a universal viewer daemon that runs on each display, a master control application that coordinates the presentation, and the UI program that controls the current position 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 [108], a programmatic interface to control applications in the Microsoft Office suite. The wrapper allows the viewer to explicitly call forward, backward, and build commands within the PowerPoint application.

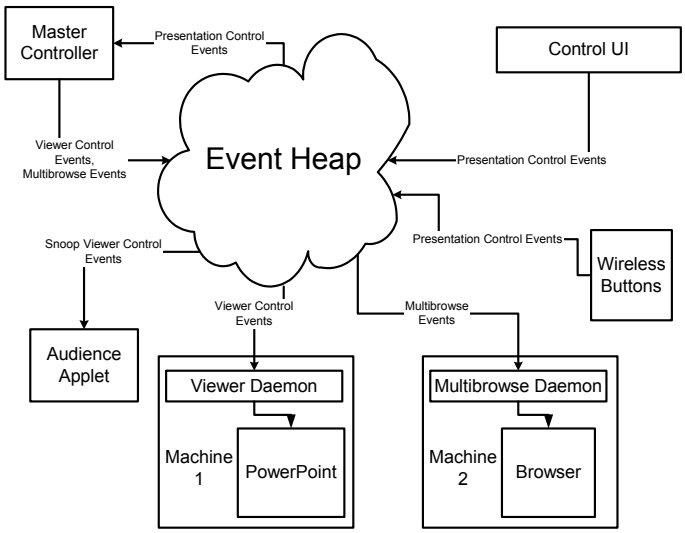


Figure 22 – SmartPresenter Paths through the Event Heap

The master control application 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. As mentioned earlier, these can be any valid Event Heap event in addition to events specifically to trigger the display of content. The master controller waits for presentation control events telling it to advance, step backward, or jump to some specific point in the presentation, and then sends the events appropriate for that point in the presentation out to the viewer applications.

The control UI that can be run in the workspace sends presentation control events out over the Event Heap to advance or reverse the presentation. These are the events picked up by the master controller that it uses to control the display of actual content in the workspace.

Tracking the current presentation for display on laptops is done by snooping on the events sent out to viewers.

Figure 22 shows how all the pieces fit together. Note that view control and multibrowse are the only type of event shown, but 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 would not take place.
- **Snooping:** Without the master presenter or any of the specific data viewers even being aware, the laptop users can 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.

Shortly after SmartPresenter was completed, the ease of coupling applications and devices through the Event Heap allowed us to extend it. We used iStuff wireless buttons and the PatchPanel (both discussed earlier in this section) 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 control UI only if they need to jump to a specific point in the presentation.

PointRight

PointRight [87] is a pointer and keyboard redirection system that operates in multi-machine, multi-user environments such as interactive workspaces. The author was the primary designer of the PointRight system. By allowing users to control the mouse and keyboard input of machines being displayed in an interactive workspace using a dedicated wireless mouse and keyboard for the space or via their own laptops, PointRight helps the support the moving control usage modality described in Chapter 3, Section 3.4.

Development of PointRight began with the simple goal of making it possible for a single mouse and keyboard to provide input to independent desktops on several large displays in a room. It was motivated by frustration with having to deal with the three or four sets of keyboards that had to be juggled to control input to the machines being displayed in the iRoom.

PointRight allows pointer control from any input device to be re-directed from screen to screen as if all of the screens were a large virtual desktop, despite their being driven by different machines.

The result is a cursor that the user moves seamlessly across the space of displays as though they were a single surface—when it reaches the edge of a screen, it continues its motion onto the connecting edge of the adjacent screen in that direction. If there is no adjacent screen, it simply stops moving and remains at the edge of the screen. If an intervening screen between two working displays is off, the cursor will skip over it. The input of any keyboard associated with the pointing device is directed to whatever machine is currently displaying the cursor.

While PointRight has been developed to work in an interactive workspace, the system will work for any networked collection of machines and input devices. To provide this flexibility it employs a general geometric model of machines, screens and the pointer transitions among them. Figure 23 gives a theoretical topology showing the various types of interconnections of screens and machines that are supported.

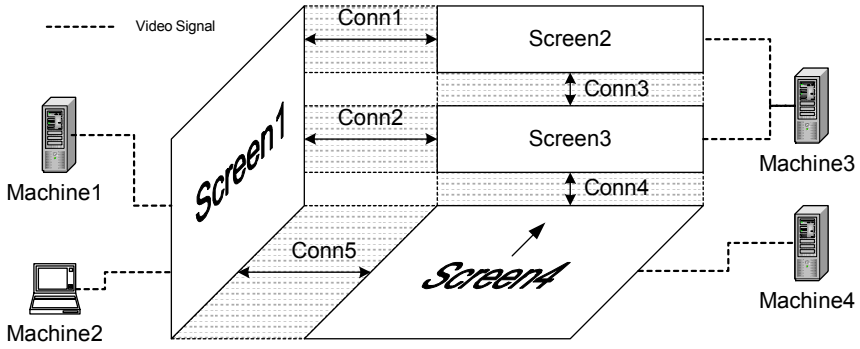


Figure 23 - PointRight Example Topology

The space topology description consists of screens, machines, and connections. As shown in the figure, there can be multiple machines displaying to a single screen (screen 1), a single machine displaying to multiple screens (machine 3 to screens 2 and 3), multiple connections off the side of a single screen (screen1 to screens 2 and 3), and connections across corners (the left edge of screen 4 to the bottom of screen 1).

Users find PointRight to provide a natural and intuitive way to interact with multiple devices and have been very positive in their assessment of its utility and usability. The fluidity of using the PointRight system is difficult to convey without seeing it in action. A video of the system can be found on the supplementary CD-ROM included with the dissertation., or is available in streaming RealVideo format at http://graphics.stanford.edu/papers/pointright-uist2002/.

PointRight uses the Event Heap for communication of which displays and machines are active, along with which machine is currently being displayed to each screen. Actual mouse and keyboard events are currently transferred over direct socket connections, although the current

version of the Event Heap has proven fast enough for mouse events and is being used in the new OS X version of PointRight.

The PointRight application consists of a *sender* that redirects mouse and keyboard events from the local input device or devices and a *receiver* accepts remote pointer and keyboard events and injects them into the local machine's event queue. Senders use the *space topology description* for the room to direct input to the appropriate display. Receivers are responsible for receiving events and rescaling cursor motions to fit their particular display. Any machine running the PointRight software can operate as either sender or receiver, but to avoid problems of recursive redirection, loops, etc., it may not be both simultaneously. The transmission of pointer and keyboard events between sender and receiver is peer-to-peer with no centralized PointRight server.

By monitoring events on the Event Heap, each sender maintains dynamic data on the state of machines and displays. Events are generated when a screen is switched on or off, a new machine becomes connected to a screen, or when the PointRight application is opened or closed on a machine. Events are posted and retrieved on the Event Heap server machine, and persist for a little more than two minutes. All active machines post events refreshing their current state every two minutes. If the senders do not see an event from a previously active machine or screen, the state of that object in the local database is updated to "off."

The PointRight system currently runs on Windows 9x/NT/2000 and on Mac OS X. An implementation is also under way for Linux. The code is available as open source [5], and there are binary installers available for Windows.

The main way that PointRight leverages the Event Heap is through its soft state mechanism. That mechanism makes it possible for machines to react to system changes without interruption. In particular, if some machine goes down, the system adapts to the change within a couple of minutes without the need for explicit intervention. The centralized nature of the Event Heap also provides a standard place to rendezvous with the displays and machines of an interactive workspace that are accessible for pointer control.

iClub

iClub is an application that turns the iRoom into an interactive dance club. It was built by four Stanford undergraduates, Yeon-Jin Lee, Xiaowei Li, Josh Samberg, and Kabir Vadera, as their senior project. Unlike other applications developed using the Event Heap it is recreational. Their ability to create iClub using the Event Heap shows that the system is useful for applications that we did not envision supporting.

The goal of iClub was to take the traditional dance club environment, which consists of techno music and synchronized computer graphics, and make it interactive. Specifically, they wanted to provide touch interaction with the graphics via the SMART Boards in the iRoom and give users the ability to control the music being played in the room. A video showing iClub in action can be found on the supplementary CD-ROM included with the dissertation. Figure 24 shows the iClub in use with some of the visualizations running in the background.

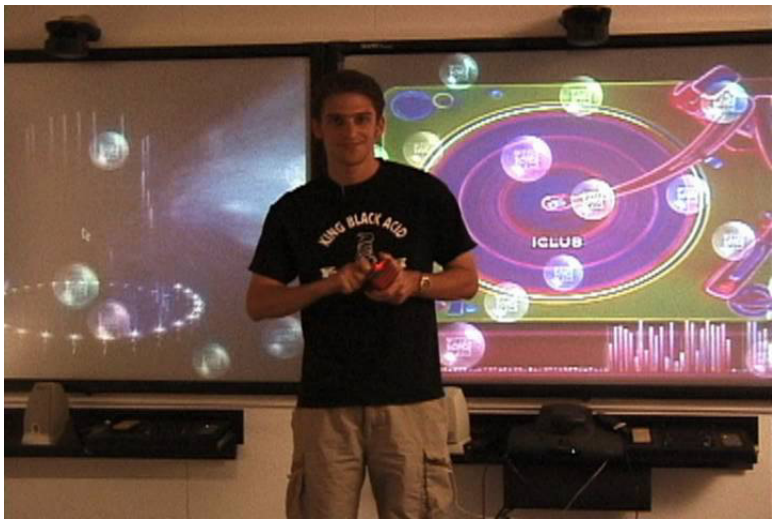


Figure 24 – The iClub in Use

The system they built has a variety of different components that all interact with one another over the Event Heap:

- **Automatic DJ:** The DJ application runs on a dedicated sound machine in the iRoom and plays back the techno songs. It posts frequency and beat information for the computer graphics applications and changes the music being played based on user interactions with the various displays and controls in the iRoom.
- **Bubbles:** Bubbles float across the three SMART Boards until a user touches them and they pop. Each bubble is associated with a randomly selected sound-effect, which is played by the automatic DJ when the bubble is popped.
- **Mermaid:** A visual object that is displayed on a portion of one of the SMART Board touch-screens in the room. The Mermaid keeps track of the total number of bubbles and if the number ever falls too low she emits new bubbles.
- **Turn-table:** Another visual object that can be displayed on one of the SMART Boards. When a user touches the record on the turn table and rotates it, the music currently being played back by the DJ speeds up or slows down depending on the direction of rotation.

- **Disco Ball:** The Disco Ball is non-interactive and runs on the interactive table in the iRoom. The coloring and effects around the disco ball change with the beat and frequency information of the song currently being played by the automatic DJ.
- **Dancing Girls:** When users touch any region of one of the three SMART Board touch-screens that doesn't contain some other object, a small dancing girl appears. The dancing girls time their dancing to the beat of the current song. After ten to twenty seconds the dancing girls fade out and disappear to avoid cluttering the displays.
- **Frequency Fountain:** Two of the displays show "frequency fountains" which pulse to the beat of the music and have virtual water jets whose heights are controlled by the frequency information for the current song. Touching the fountains causes flashes of lightning on the screen.
- **Voting Station:** Laptops with a touch sensitive displays can serve as voting stations. Each voting station displays the names of several songs and users can select the one they prefer to hear next.
- **Frequency Filter:** This is another application that can run on laptops with touch displays around the iRoom. It displays a single slider which controls a low pass filter. As users move the slider down, higher frequencies are gradually removed from the music that is being played back by the automatic DJ.

The components of the iClub were all built as stand-alone applications that emit and react to various different events using the Event Heap. Here are the event types used by iClub and the mechanisms that they enabled:

- **Beat:** Beat events are sent out by the automatic DJ every time there is a beat in the current song, which occurs up to 170 times per minute for techno songs. The frequency fountains, dancing girls, and disco balls poll for these events and adjust the graphics they are displaying. The distribution paths for beat events is shown in Figure 25.
- **Frequency:** Frequency events are sent out by the automatic DJ about once per second. They contain a breakdown of the power in thirty-two different frequency bands for the section of the song that is currently playing. These events are picked up by the frequency fountain and used to adjust the heights of the virtual water jets, and are also picked up by the disco ball.

- **Bubble Pop:** Each time a user pops a bubble, the graphical display application sends a bubble pop event containing the name of the sound effect to be played. The automatic DJ polls for these events and plays back the appropriate sound.
- **Bubble Move:** When a bubble hits the edge of its current screen, the graphical display application sends a bubble move event indicating the side, coordinate and velocity vector of the exit. Graphical display applications also poll for bubble move events and begin displaying bubbles that have exited adjacent screens using the parameters specified in the bubble move event.
- **Music Speed:** The turn-table graphical display application emits music speed events indicating that the music should play faster or slower based on user interactions. The automatic DJ listens for music speed events and speeds up or slows down the music accordingly.
- **Voting Events:** Whenever a user votes for a song using the voting application, it emits a voting event with the name of the song. The automatic DJ picks up the voting events from the various voting applications in the space and tallies them to select which song to play next.
- **Frequency Filter:** As users interact with the frequency filter application, it emits frequency filter events indicating how to set the low pass filter being applied to the music. The automatic DJ picks up these events and filters the music appropriately. Figure 25 shows the distribution paths for frequency filter events (including another frequency filter mechanism using an iSlider which will be discussed later)

The entire iClub suite was implemented over the course of about a month and a half by the four authors who had never met before (they were paired together in the senior project class). Using the Event Heap gave them flexibility in writing their application. Early on they determined the set of events to use, and then parceled out the various components among themselves for implementation. Integration took place over the last week and a half, and required few changes to the Event Heap portions of the code in the various components.

The iClub is a good example of the importance of the need for failure tolerance in the Event Heap. Since the individual components were developed quickly by a variety of different people, during integration most of the components had bugs and would frequently crash. The decoupling in the Event Heap allowed the other components to continue running, however, and the author of

the crashed component was able to modify his code and restart the component without having to shut down any of the other parts of the iClub.

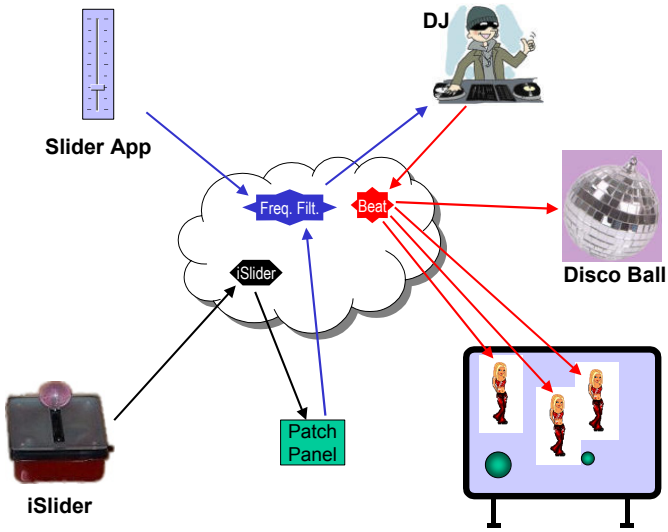


Figure 25 – Event Paths for the iClub
(Black = iSlider Events, Blue = Frequency Filter Events; Red = Beat Events)

At the end of the iClub's development, the authors were able to modify its functionality using the extensibility provided by the Event Heap. Specifically, they used an iSlider, which is part of the iStuff collection, and the PatchPanel (both of which were discussed earlier in this chapter) to provide another mechanism for controlling the frequency filter. Figure 25 shows the paths for frequency filter events, both from the original slider application and from the iSlider. The PatchPanel was configured so that iSlider events would be transformed to frequency filter events. Since the automatic DJ was already configured to pick up frequency filter events, it was able to run unmodified using the iSlider for input. Integrating the iSlider with the iClub took less than an hour of work.

Applications Summary

As was just shown, the Event Heap is capable of supporting a wide range of different applications. It is also important to try and gauge how much extra work is required to develop an Event Heap enabled application beyond what it would take to develop a normal stand-alone application. Unfortunately, since most of the applications described were developed independently, we do not have detailed measures of lines of code, total programmer hours or other specific measurements of the difficulty. We present here a qualitative description of

approximate difficulty for some of the applications built on top of the Event Heap, and, where available, we give details on lines of code or development time:

- **CIFE Suite:** About 100 lines of code were required to enable their legacy '4D-Viewer' application to both send out Event Heap events on state change and respond to events sent by other applications. For other applications they have reported that the overhead for developing the Event-Heap-specific code is low.
- **Multibrowsing:** The initial version of multibrowsing took a graduate student (the author) less than one day to implement. Developing the multibrowse-2-go plug-in for Internet Explorer took an undergraduate one summer to implement, but much of that time was spent on the Internet Explorer plug-in interface.
- **SmartPresenter:** SmartPresenter is 993 lines of code long for all of its programs (individual viewer daemons, master coordinator, GUI controller, etc.), but only 40 lines are related to Event Heap coordination. The PowerPoint wrapper took a graduate student about one day to implement after they learned how the Office Automation mechanism functioned (which took about a week).
- **PointRight:** The initial version of PointRight was implemented over a period of about one month by the author, but only a few days of that time were spent on the code to handle state coordination using the Event Heap.
- **iClub:** iClub was written over a period of about one month by four undergraduates who had not met each other before working on the project. Components were developed independently, and integration took less than one week. Integration with iStuff to provide extra mechanisms of control was done over the period of only a few days towards the end of the development process.

Overall, developers have experienced a low overhead both for adding Event Heap code to their applications and in integrating their applications to coordinate with others over the Event Heap.

Finally, although throughout the section we presented what we called "applications," since the components were constructed using the Event Heap, they can all be made to flexibly interoperate with one another. For example, SmartPresenter could be set up to control the applications in the CIFE Suite during a presentation. This was part of the goal of the Event Heap model: to provide users with the capability of integrating diverse collections of applications to address the problem at hand.

6.1.2 Deployed Environments

Since the public release of the system, iROS, and by extension the Event Heap, has been deployed in a variety of locations both on Stanford Campus and around the world, including many intended for non-Computer-Science related work. We describe below some of the environments that have been deployed. For each one we give an overview of the interactive workspace, a description of the main applications and iROS systems used, and a discussion of how the Event Heap model has been effective for that environment. Although the deployments have not been perfect (we describe some problems below), iROS has been sturdy under a variety of conditions of use by people other than its creators. Table 13 gives a list of some of the environments where iROS and the Event Heap have been deployed.

Deployment	Location	Description
The iRoom	Gates Hall, Stanford Campus	The original prototype interactive workspace.
IRoom-to-Go	CIFE, Stanford Campus	A three SMART Board, front projected workspace that can be set up in a construction trailer.
Flex Lab	Meyer Library, Stanford Campus	Classroom for writing courses.
SLL Prototype	Vining Hall, Stanford Campus	Prototyping environment for investigation different learning scenarios in interactive workspaces.
iLoft	Stanford Campus	Interactive workspace for researching technologically enhanced collaborative design work.
SCIL Classrooms	Wallenberg Hall, Stanford Campus	New high-tech classrooms designed to test the use of cutting edge technology in education.
jRoom	Stanford Campus	A personal iRoom used by a developer to test iRoom applications in his own cubicle.
iLounge	KTH, Sweden	Prototype interactive workspace for our collaborators in Sweden.
L3S	Hanover, Germany	Prototypes for classrooms that support local students and remote learners.

Table 13 – Some of the Deployed iROS/Event Heap Environments

Original iRoom

The iRoom in Gates Hall B23 on Stanford Campus was our original prototype interactive workspace. It was discussed in detail in Chapter 1, Section 1.2.1. An overview image of the current version of the iRoom with an inset of the high-resolution interactive mural display is shown in Figure 26.

The iRoom is regularly used for the group meetings of the Interactive Workspaces Project. During those meetings, the PointRight system and Multibrowsing are regularly used.

PointRight is most commonly used by people to interact with the machines displaying to the SMART Boards without having to get up and walk over to the boards. This frequently happens

when someone is presenting at the boards and an audience member just wants to briefly access the machines to demonstrate something.

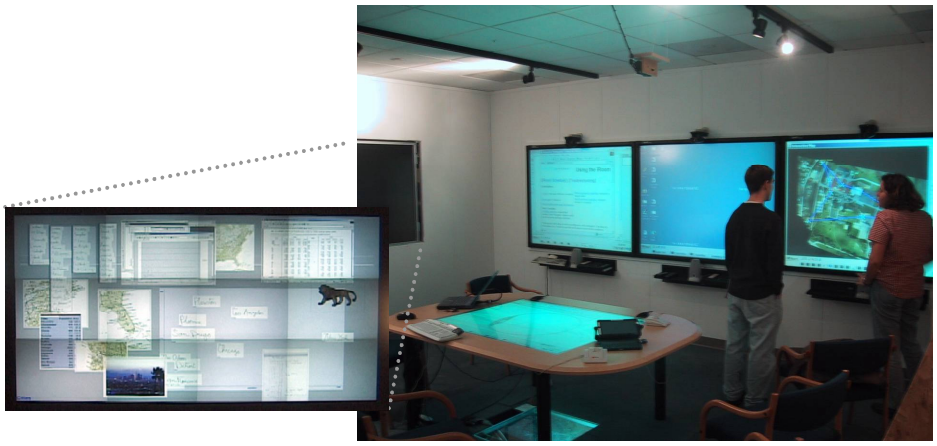


Figure 26 - The iRoom with Inset of Interactive Mural

Multibrowsing is regularly used in two different ways. First, it is used to open content linked from a web page on one of the SMART Boards on a different SMART Board. Second, it is used by group members to push information that they found during private browsing on their laptops onto one of the public displays.

The iRoom is also frequently used to give demos to visitors. During these demos, most of the applications described in the previous section are shown to the visitors.

Finally, user studies are often staged in the iRoom. For example, the Workspace Navigator application, which allows the recording and playback of user sessions, was recently tested in the iRoom.

As the test-bed environment for the Interactive Workspaces Project, the iRoom deployment stress tests the Event Heap implementation the most. In practice, the current implementation of the Event Heap stays up and running continuously with reasonable performance. The server typically needs to be restarted every few weeks, and the client machines are restarted about once per week. Earlier versions of the Event Heap were considerably less stable, frequently locking up or providing extremely long latency (on the order of 5-10 s). During the deployment of those infrastructure versions the iRoom was not very usable: it was meaningfully active only for demos.

iRoom-to-Go (CIFE)

The researchers at the Center for Integrated Facilities Engineering (CIFE) are the authors of the CIFE Application Suite discussed in the applications section (Section 6.1.1). They are investigating the use of interactive workspaces to improve planning and managing large construction projects. To run their application suite they have created what they call "iRoom-to-Go." It consists of a set of three front-projected SMART Board touch screens which are fairly easy to transport. The idea is that their system could be packed up and set up in a construction trailer with minimal difficulty. Their setup is shown in Figure 27.

Figure 27 - CIFE's iRoom-to-Go

The iRoom-to-go setup uses PointRight to provide a single mouse and keyboard to control the three machines that display in their environment. Beyond that, they of course use the CIFE Application Suite, which was described in detail earlier.

Overall, the Event Heap has worked well in the iRoom-to-Go installation. The Event Heap has allowed them to build coordination into their collection of legacy and custom applications. They had some problems with latency and lock ups with early versions of the Event Heap. They also experienced a problem with poor application performance when using non-blocking calls to poll the Event Heap server. This was fixed when the application code was changed to use a blocking poll.

FlexLab Writing Laboratory

The Program in Writing and Rhetoric (PWR) deployed iROS on an experimental basis in the FlexLab at Meyer Library on Stanford campus to prototype teaching strategies for collaborative writing. This was done for actual writing classes during the 2001-2002 school year. The room consists of five stations, each with a computer that displays to a large plasma screen. Each station also has three laptop computers where individual students can work. A final computer drives a large rear-projected display at the front of the room where work can be displayed to all students.

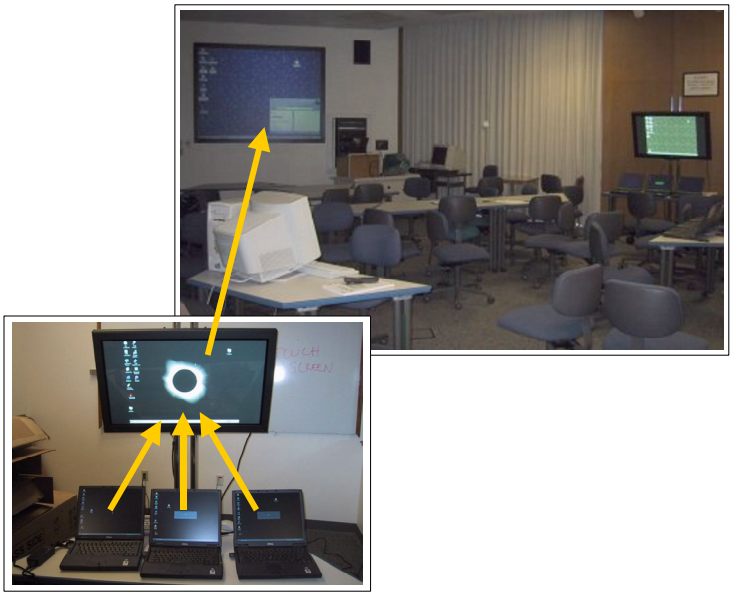


Figure 28 - The Flex Lab

The PWR researchers and teachers have primarily used PointRight to provide access among the displays in the FlexLab. They have configured it such that students can mouse off the top of their laptop to control the computer driving the plasma screen, and mouse off the top of the plasma screen to control the large display at the front of the room. This allows students to demonstrate their writing either to their local cluster or to the entire class. The PointRight connections and the FlexLab itself are shown in Figure 28.

The PWR researchers and teachers had mixed results with the system. While some classes used the system, others decided not to do so. Further, the system itself did not perform reliably, with client computers crashing fairly frequently. Unfortunately, the exact cause was not isolated, but it appears that their problems were related to device driver configuration on the laptop PCs.

Stanford Learning Lab Vining Facility

The Stanford Learning Lab (SLL, which has now become the Stanford Center for Innovation in Learning, SCIL) has deployed an interactive workspace in their Laboratory in Vining Hall. . It is constructed out of foam core and is designed to be quickly rearranged to test different physical space configurations. They use the space to run user studies on how learning can be facilitated in interactive workspaces. The primary setup has two displays front projected onto foam-core and a table where three test computers can be setup for use by participants in their studies. The Vining Lab setup is shown in Figure 29.

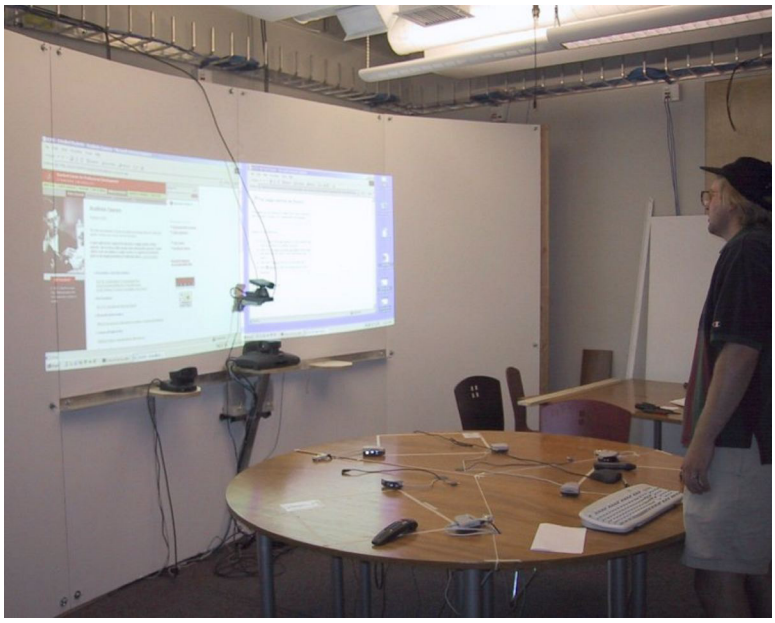


Figure 29 - The Stanford Learning Lab Test Facility

Since the lab displays are not touch screens, the only input to their system is using the PointRight system from a dedicated wireless mouse and keyboard or from individual laptops. They also use a collaborative brainstorming tool called GroupBoard, which was developed by Jeff Raymakers and Hans Anderson, to do evaluations of students working on team projects in the prototype space.

Aside from PointRight, the researchers at SLL rely on the failure tolerance provided by the Event Heap to insure that a crash in one of their test stations doesn't interfere with other study participants. They also rely on the transparent communication of the Event Heap to log the events triggered by the interaction of their test subjects with the Event Heap connected test computers. They have reported the occasional need to reboot systems, but overall the Event Heap system has been stable for them.

Wallenberg Hall Classrooms

Wallenberg Hall is a newly renovated building on Stanford Campus, which just opened in the fall of 2002. It is designed to be a test-bed for the use of technology to facilitate learning. Four of the classrooms in Wallenberg Hall are set up to use the iROS infrastructure. These classrooms have SMART Boards on rollers at the front of the room and allow students to participate electronically in the class over an 802.11b wireless network from laptops. Currently the classrooms are set up to use PointRight and Multibrowsing. Researchers helping to support classes in the building are interested in building other applications on top of the Event Heap, but since the building just opened, no development has yet occurred. For the same reason, it is too early to evaluate the success of the Event Heap and iROS in these environments. A picture of one of the Wallenberg Hall classrooms is shown in Figure 30.

Figure 30 – The Room 120 Classroom in Wallenberg Hall

iLoft

The iLoft is located in the Center for Design Research (CDR), which is also known as Building 560, on Stanford Campus. It is used by researchers in the Mechanical Engineering Department to investigate how technology can facilitate collaborative design. Before they built the iLoft, the researchers in the CDR ran initial studies using the Stanford Learning Labs Vining Facility (discussed earlier). A picture of the iLoft is shown in Figure 31.

As the figure shows, the iLoft looks like a non-technologically-augmented open work area. They have constructed portable touch screen units with back projected 3'x4' screens which they call

"LightBoxes" for use in the space. These units can be rolled into the space and positioned by teams doing collaborative design. Standard computers with LCD monitors can also be set up on tables in the space so that individual users can have a private workspace. The iLoft also supports 802.11b networking so that users can bring in laptops with wireless support.

Figure 31 - The iLoft

The iLoft runs a complete installation of the iROS system including PointRight and Multibrowsing. The latter is used during collaboration sessions to move web content among the laptops, PCs, and LightBox systems in the space. They also use the GroupBoard system for collaborative brainstorming that was mentioned in the discussion of SLLs Vining Facility.

In the studies the CDR researchers have done, the failure tolerance and transparent communication features of the Event Heap have been important. During user studies, the systems failure tolerance has allowed them to fix or reboot systems that have crashed without interruption to other participants. Transparent communication allows them to log the event traffic generated by the participants during their activities. These logs are later analyzed to determine how effective the tools were in making collaborative design process more efficient.

The CDR researchers are also conducting studies in another loft, which is near the iLoft, that is used by the ME310 product design class. Students in that class use an application built on the Event Heap called WorkspaceNavigator, which was developed by Arna Ionescu [82]. Workspace Navigator periodically captures the state of the workspace during collaboration sessions and allows students to later search back for what content was being displayed in the interactive

workspace at any particular time. Students record their initial product design brainstorming sessions for the class and are then able to reference that material in future sessions.

jRoom: iRoom in a Cubicle

One of the Interactive Workspaces developers, Brian Luehrs, has deployed a personal interactive workspace in his cubicle, which he calls the "jRoom." It is shown in Figure 32.

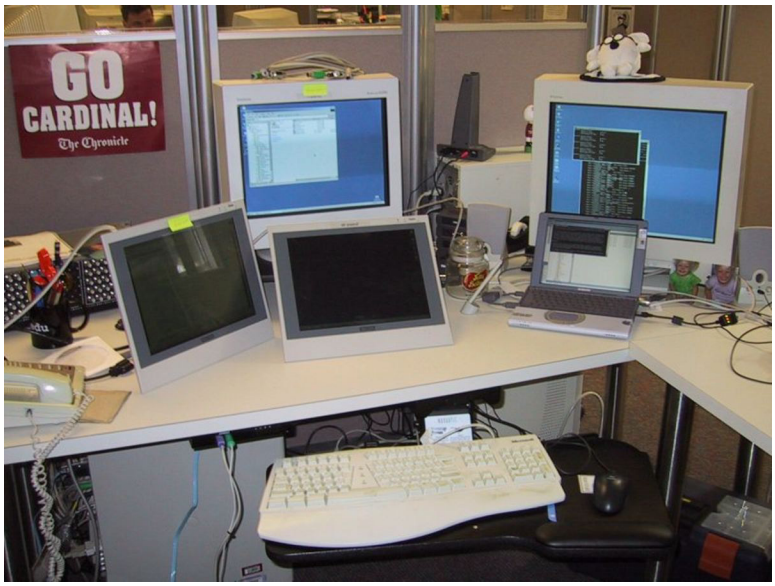


Figure 32 - The jRoom Setup

He uses PointRight to allow himself to switch back and forth between the various displayed machines without having to switch keyboards. He also uses the setup to write and test applications that may be deployed in the Wallenberg Hall classrooms. The jRoom deployment demonstrates that although the Event Heap and the basic applications that come with the iROS are not intended to be used in this domain, they can also be useful when deployed in personal spaces.

iLounge

The iLounge is an interactive workspace in Sweden at the KTH Kista Forum building. It is being used as part of the iSpace Project, a Wallenberg Global Learning Network (WGLN) sponsored collaboration between researchers at Stanford and researchers in Sweden. The researchers at KTH intend to use the iLounge primarily for collaborative design work, but they also hope to use it for their own group meetings and for student project work.

The iLounge space itself is set up to be very configurable with lighting that can be changed and walls that are designed to allow the easy mounting of various equipment. It has two rear-

projected touch-sensitive SMART Boards and a table with an embedded touch-sensitive plasma screen. Another corner of the room is designed for informal work in smaller groups. It has a 42" plasma screen displaying a PC (which is controlled by a wireless mouse and keyboard), a small table, and three easy chairs. The space also has wireless LAN support and a sound system. A picture of the iLounge is shown in Figure 33.

Figure 33 - The iLounge

The researchers at KTH have installed the entire set of iROS applications in the iLounge (the Event Heap, Data Heap, iCrafter, PointRight, and Multibrowsing). They are also running some of their own infrastructure for ubiquitous computing. Their space was constructed in the spring of 2002, so it is relatively new and they are still trying to determine how best to use it.

Since the iLounge will serve a purpose for the Swedish researchers similar to what the iRoom serves for us, we expect it to get a lot of use. One interesting software component they have already designed is an Event Heap bridge that selectively forward events between two interactive workspaces each with their own Event Heap. We hope to conduct collaborative sessions in the iLounge and iRoom in the future using this technology.

6.2 User Survey

As the previous section showed, the Event Heap has been widely deployed and it has been used in a variety of applications over the past three years. While this result is encouraging, feedback from users is also desirable. To that end, we conducted an informal survey of users of the Event Heap in the fall of 2002. It was divided into two parts, with one section for developers, and the other for administrators. The survey was announced on several email lists related to either the

Interactive Workspaces Project at Stanford or the Open Source release of the iROS system. Users were asked to self-select as either developers or administrators when taking the web based survey. The survey itself was given using the SALGains [132] web site. After all participants had filled in the survey, the website provided percentage breakdowns for each question.

The remainder of this section gives a summary of the most interesting results from the survey. Overall, twelve developers and three administrators filled out the survey. The small sample size for the administrator survey makes statistical results from it invalid, so only comments from that survey's participants are reported here. Results from the developer survey, however, include statistical break-downs where appropriate. The survey questions, compiled results, and raw survey data can all be found in the survey appendix, which is found on the supplementary CD-ROM included with the dissertation.

Some of the questions on the survey were designed to gauge whether the participants agreed that the characteristics of interactive workspaces and the properties and features of the Event Heap model that we posited were valid. Other questions asked how well participants believed the Event Heap implementation provided for the characteristics, properties, and features. The survey was informal and not designed to be thorough in its coverage of these items. It therefore did not contain questions for each characteristic, property or feature.

6.2.1 Characteristics of Interactive Workspaces

Both developers and administrators were asked about the use of application ensembles in interactive workspaces. We had reasoned that most applications for interactive workspaces would be ensemble based, rather than monolithic, due to the human centered interaction and flexible reconfiguration characteristic (**H2**). Ninety-two percent of developers and all three administrators agreed with us, seeing application ensembles with an occasional monolithic application as being the most common case.

Eighty-three percent of developers agreed that Interactive Workspaces are likely to be dynamic on short time scales (**T3**) due both to failure and also to portable devices entering and leaving. Another respondent said:

> "I think [interactive workspaces] will eventually become dynamic on short time scales (where short is on the order of an hour or so). I think that's the right time scale because that's how long meetings tend to last. Once everyone is in the room, they tend to stay for a while, and the system is pretty stable for the duration of the meeting. Device failures, etc occur, but I don't think this will occur often enough for change to occur on the order of minutes."

An open-ended question asked developers and administrators what sorts of platforms and standards they thought the Event Heap should support. Their suggestions and the breakdown of how many people suggested each standard or platform are listed in Table 14. Since participants were allowed to list as many platforms and standards as they wished, the sum of the recommendations for each standard or platform is greater than the number of participants. The developers and administrators wanted a diverse set of platforms to be supported, which helps validate our assertion that interactive workspaces systems infrastructures will need to provide for heterogeneous software platforms (**T2**).

Standard	Developers Recommending (Out of 9)	Administrators Recommending (Out of 3)
C++*	7	2
Java*	6	2
HTTP*	2	2
Visual Basic	2	0
C	1	0
Cocoa and Objective-C	1	1
Perl	1	0
.NET	0	1
ZeroConf	0	1

* = Already supported.

Table 14 – Platforms Recommended for Support

Overall, C++ and Java were considered the most important platforms to support by both developers and administrators. This is a favorable result since both of these platforms are already supported (although the current C++ implementation is one revision behind the Java version). One developer made the comment that supporting a pure C implementation would probably be of greatest utility since most other languages support bindings to C libraries. A C implementation could therefore be leveraged to provide support in other languages. The current C++ implementation is itself embedding Java and is therefore not a good candidate for being embedded itself in other languages.

Overall, survey participants concurred with us on the nature of interactive workspaces. Specifically, they believed that human centered interaction with application ensembles, dynamic change, and heterogeneity were all important aspects of interactive workspaces.

6.2.2 Coordination Infrastructures Properties

One of the properties we proposed is extensibility (**P3**). Developers saw this as being important, with seventy-three percent foreseeing themselves adapting their applications to work with other programs in the future. One respondent added the following comment about adaptation:

> "Almost every app I wrote for the iRoom had to be adapted later for some other purpose, whether I planned it or not. It became very important to think about extensibility before writing new apps, so it would be easier to adapt when (not if) necessary."

Another proposed property was expressiveness (**P4**). Developers were asked if the Event Heap had been sufficient for expressing the coordination they used in their applications. Seventy percent of developers agreed or strongly agreed that the Event Heap had been sufficient. Only one developer disagreed, saying:

> "While the EventHeap [sic] is good for many coordination tasks, there are some tasks it does not handle. Most notable are bulk data transfer and persistent state. There are other iROS APIs to handle some of these tasks, and how well they handle them is not what this survey is about - however, the point is that the EventHeap [sic] cannot stand alone."

This comment is mostly reasonable, although the persistent state service to which the respondent referred was built on top of the Event Heap, showing that the Event Heap is capable of expressing that type of coordination. The Event Heap is not designed for bulk data transfer, as will be discussed in Chapter 7, Section 7.4.

The simple and portable client API property (**P5**) was included in part to simplify the learning curve for those using the Event Heap. Ninety percent of developers agreed or strongly agreed that it was easy to adapt their previous programming techniques for use with the Event Heap. The one person who disagreed said:

> "Programming for the EventHeap [sic] involves thinking about problems that don't occur in other environments, but this has more to do with the applications being written than the EventHeap [sic] API. For example, the possibility of multiple users interacting with an application at the same time is an issue with EventHeap [sic] applications, because it is possible, while it generally isn't with a normal application."

In another question, developers were asked how easy it was to develop with the Event Heap. Seventy-two percent of developers felt that it was easier and quicker to develop stable applications and application sets using the Event Heap than it had been with previous systems they had used.

One developer offered the following anecdote about how easy it was to integrate a component he had written with one made by another programmer:

> "I built a system to send user barcodes from the control panel machine [a computer in the iRoom] to a smart board [sic] in the iRoom. I developed the barcode entry system and another person developed the UI component. Our applications only had to know what the other expected in terms of fields being

> sent. Other than that we developed our code independently and the first time we put it together it worked. It was really amazing to see it work for the first time."

To see how well we had done with the easy debugging property (**P6**), developers were asked how easy it was to debug Event Heap applications compared to other applications they had written. Fifty-five percent of developers agreed that it was very easy to debug Event Heap applications compared to others they had written, while 27% disagreed. Since distributed applications like those that use the Event Heap are typically more difficult to write and debug, it is encouraging that respondents on average found it easier to debug Event Heap applications than other applications they had developed, including non-distributed ones.

Since perceptual instantaneity (**P7**) was already known to be a critical property, respondents were not asked whether they considered it important. Instead, developers were asked how well the Event Heap implementation performed. These question combined throughput and latency, even though only the latter relates to perceptual instantaneity. Fifty-five percent of developers found the performance of the Event Heap to be better than they expected, 36% found it acceptable, and 9% (one person) found it only tolerable. Comments suggested that developers generally believe that while the system is pretty good, there is still room for improvement.

One administrator made the following comment that supports our decision to have perceptual instantaneity as an important property:

> "The original event queue model was a step in OS architecture that was mainly aiming at improving the support for interactive applications. It now appears as if iROS could play a similar role in post-desktop, ubicomp environments. Therefore, take user interface development needs (latency etc.) into account when further developing iROS."

6.2.3 Event Heap Model Features

In comments on the question about content-based routing (**F1**), two of the respondents indicated that they usually only used the 'EventType' field for content-based routing. This indicates usage of multi-cast distribution within a group of applications sharing the same event types. Given this, it might be possible to restrict the number of fields used in matching and routing to speed up Event Heap operation. This change could be made without changing event transparency, thus improving performance while maintaining some of the desirable properties associated with transparency such as extensibility (**P3**) and easy debugging (**P6**).

The Event Heap was designed to support all of the major routing patterns (**F2**). Developers were asked how frequently they used each of the four major communication patterns, point-to-point, multicast, broadcast, and anycast. Multicast and broadcast were the most commonly used, with

about one-third of respondents indicating that they use these two types "very often." All of the others were used at least sometimes by around one third of the respondents, however, confirming the need to support all of the different routing patterns.

The standard routing fields feature (**F3**) appeared to be not very well understood by developers. One developer noted in the survey that, aside from 'EventType' and 'TimeToLive,' they don't use the default fields very often. Based on casual conversations with developers, this seems to be a fairly widespread sentiment. Aside from the two mentioned fields, all the other fields are for routing, so routing is either not useful, or not well understood (the latter is quite possible since the routing fields were only recently added to the Event Heap).

One surprising result of the survey was that query persistence/registration (**F5**) was regarded as quite important. By a margin of eleven-to-one, the developers that took the survey felt that push-based event reception (i.e. registration or subscription) was a better fit for their application than pull-based reception (i.e. polling), which is the default tuplespace mechanism. One respondent commented:

> "Using registration and callbacks has the advantage of not blocking the listener. Quite often, an application wants to listen for certain kinds of messages for the lifetime of the applications, so registration is easiest."

While pull-based event delivery will always be needed for anycast routing and query-type access to the Event Heap, it seems that future performance-enhancement work should be focused on the registration mechanism.

The flexible typing feature (**F7**) consists both of adding a type to tuples and allowing extra fields to be added to tuples of a given type without breaking their ability to match other tuples without the extra information. We asked developers how useful the extensibility aspect had been. Ninety-one percent of developers agreed or strongly agreed that the extensibility had been useful, with nobody thinking that this feature had not been useful. One developer found this capability very helpful, saying, "I used this extensively to add extra information that receivers need."

Event Heap events are a simple collection of fields. One concern was whether this format was sufficient for developers, who might have wanted support for more complex objects. Ninety-one percent of developers, however, agreed or strongly agreed that the event structure was sufficient for their application development needs. One respondent commented that it would be nice if there were support for hierarchical data structures in events to save developers the time of flattening complex objects down to strings. Only one person made any comment about needing

to transmit first-class language level objects, however. Based on these responses it appears that the current event format is sufficient for developer needs.

6.2.4 Reliability and Stability

To assess how well we had done in implementing the Event Heap, we asked developers and administrators to what extent they agreed with the statement "the Event Heap is very reliable and stable," . Fifty-five percent of developers agreed that the current Event Heap implementation was very reliable and stable, one person disagreed, but didn't provide any explanatory comment, and the others were neutral. The C++ client was noted as being problematic, which is understandable since there is no pure native C++ client—the current version uses JNI and instantiates its own JVM to run the Java Event Heap client inside the C++ application. A few respondents noted that under high loads there have been some problems.

In contrast to developers, of whom only a little more than fifty percent felt the system was stable and reliable, all three administrators agreed or strongly agreed that the Event Heap and iROS system were stable and reliable. There are several possible reasons for the difference of opinion between developers and administrators. One possibility is that developers have more first hand experience with the Event Heap system, and therefore have a more accurate sense of its reliability and stability. Another possibility is that developers see more reliability problems, but those are being caused by bugs in their own code or other libraries they use, while finished applications seen by administrators running in their interactive workspaces are stable and reliable. A final possibility, related to the previous one, is that developers work around bugs in the Event Heap implementation during development so that the deployed applications are reliable and stable.

In any case, while Event Heap v2 goes a long way towards providing a reliable and stable implementation, especially in comparison to previous versions, there is still room for improvement.

6.3 Performance Measurements

One of the characteristics of interactive workspaces described in Chapter 3 is human level performance needs (**H3**). To summarize, this means that software in an interactive workspace must have a low enough latency that humans interacting with the software don't notice or get annoyed by delay during caused by application coordination. This motivated the Event Heap model property of perceptual instantaneity (**P7**). As discussed in Chapter 4, Section 4.1.7, for the actions we are supporting this specifically means that the effects of actions triggered by a user should be perceived within 100 ms.

Unlike with the other features of the model which are designed into the implementation (such as transparent communication), perceptual instantaneity is a function of the final system design and the traffic loads that the system will encounter. While it might be possible to model or simulate the latency, the system is sufficiently complex that is simpler to measure the performance of the actual implementation. This section presents performance measurements of the Event Heap implementation showing that it is more than fast enough to satisfy the perceptual instantaneity property.

6.3.1 Event Heap v1 Performance

As mentioned in the implementation chapter (Chapter 5), the original versions of the Event Heap were implemented on top of the T Spaces system from IBM [145]. Those versions both exhibited inconsistent behavior, with latencies sometimes increasing to several seconds, and frequent server lockups. During normal operation, the performance was usually adequate. This section presents some basic numbers for the Event Heap v1 system as a point of comparison with Event Heap v2. These measurements were made by Shankar Ponnekanti, a member of the infrastructure sub-group of the Interactive Workspaces Project.

The scaling performance of the Event Heap was measured by observing the latency of the Event Heap under varying loads. The load on the Event Heap was controlled by running a number of client processes on a cluster that accessed the Event Heap in various ways (the clients performed submission, retrieval and removal of events). The T Spaces server was configured to use 256MB of physical memory and run on a quad-processor Pentium III 550 MHz Linux server. With no load on the T Spaces server, we measured a latency of 30 ms. As the load increased, however, we noticed a wide variation in the latency depending on several factors such as the duration for which the T Spaces server had been running, the number of tuples in the server, configuration settings for the T Spaces server, etc. For example, with 200 clients, latency varied from 100ms to 2 seconds.

6.3.2 Event Heap v2 Performance

Event Heap v2 has proven to be much more stable than v1. While there are still occasional reports of lockups or high event latencies, they have been too infrequent to be able to track down potential causes.

This section presents the more detailed measurements of the performance of the Event Heap v2 implementation. Emre Kıcıman assisted with taking the measurements presented in this section and plotted the graphs.

Experimental Setup

We tested the performance on a cluster of 31 Linux workstations. Each machine was a dual-Pentium III 800 MHz with 256 MB of RAM with the exception of the server, which had 512 MB of RAM. Clients were distributed as evenly as possible across the client machines.

In a real life deployment, latency and throughput are affected by both background network traffic on the sub-net or sub-nets being used by the Event Heap server and the client applications. Since the nature of what this background traffic might look like is unknown, the machines we used in our testing were connected together on a private sub-net to minimize interference from other traffic on the network. Thus, the numbers presented represent the minimum latencies and maximum throughputs that the current implementation can provide in the absence of any other network traffic. To determine the latency for an actual deployment, the round trip latency overhead to the Event Heap server given the network load for that interactive workspace must be added to the latency numbers presented here. Estimating throughputs for actual deployments is slightly more complicated since the performance of the Event Heap server machine will determine whether the server implementation or local network bandwidth is the bottleneck for any given interactive workspace.

Three different types of measurements were made during the test runs. The first measurement was the latency for a client application to post an event and then retrieve the same event back from the server.

The second measurement was throughput in terms of the number of requests per second being handled by the server. To make this measurement, feedback was required from the Event Heap server application. A command line switch on the server triggers it to post an event containing the average requests per second along with several other statistics once every ten seconds (the average is always over the preceding ten second interval). The period of posting compared to the average latencies is long enough that its impact on the measurements is presumed to be negligible.

The final measurement was the time until all clients reconnect after a server restart. This was measured using additional data encoded in the aforementioned server statistic event. One of the statistics in that event is the number of currently connected clients and the time in ms at which the last client connected. Another is the time in ms when the server finished coming up. The test procedure then went as follows:

1. Some number of clients, N, were started and connected to the server.

2. The server was killed and restarted.
3. A client monitoring the server statistic events waited until the number of connected clients reported by the server was equal to the original number of started clients, N.
4. The time when the server came up was subtracted from the time of the last client connect to determine the number of ms until all clients had reconnected.

A percentage of total clients connected at any given point in time was also calculated using a similar method and adjusting the rate of server statistic events to be more frequent.

Latency vs. Number of Clients

Our first test measured the change in latency as more clients were connected to the Event Heap server with a constant traffic of 100 events per second distributed evenly across all the clients (except the client that measured the latency which performed the probe every second). Each of the clients used a different event type. The results of the test are shown in Figure 34.

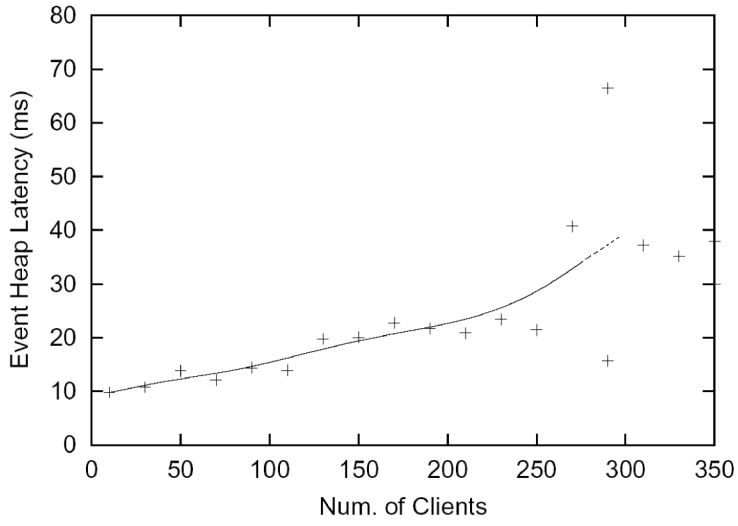


Figure 34 - Event Heap v2 Latency vs. Number of Clients

As the figure shows, the latency steadily increases from 10 ms to around 20 ms at 250 clients. After that the variability in latency greatly increases, but in no case does it exceed 70 ms. We suspect that the variability arises from thread swapping issues in the Event Heap server JVM since the server implementation is using one thread per client connection. The test could only be run out to 350 clients since the JVM would crash having run out of threads to allocate to new connections after that point

Latency vs. Server Throughput

The second test measured the latency of the system as the total requests per second being generated across 100 clients was increased. The test was run two times, once with all clients using different event types, and another time with all using the same event type. This potentially affects the result since the server hashes by event type. The latter case also reflects the potential situation when many clients in the space are all beaconing using the same event type. The results of the test are shown in Figure 35.

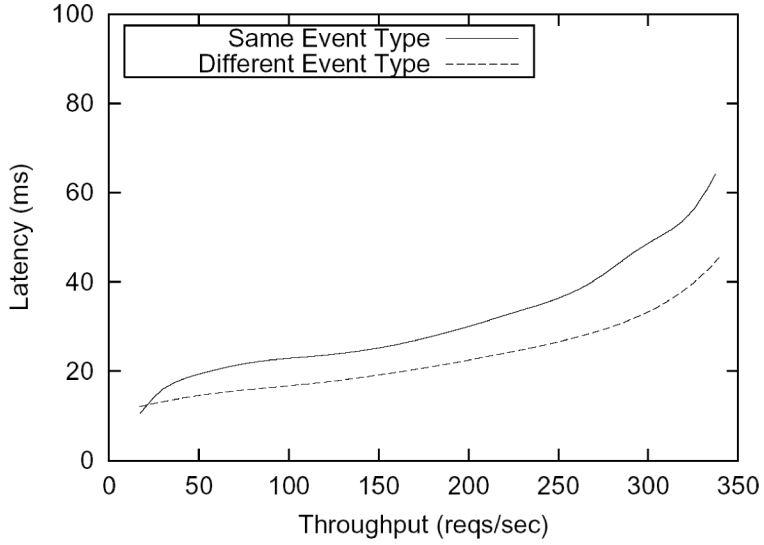


Figure 35 – Event Heap Latency vs. Server Throughput

This test shows that the latency increases gradually with throughput until a server load of about 250 requests per second is reached, rising from about 15 ms to 20 ms when clients all use different event types. As might be expected, when clients all use the same event type the latency is slightly greater at all throughput levels, increasing to around 35 ms of latency at 250 requests/s. After 250 requests/s, the latency starts to go up more sharply for both types of event traffic, but even by 350 requests/s it only goes up to around 60 ms.

Server Reconnect Time

Another concern is how much of a disruption is caused by a crash in the Event Heap server. The server itself is run inside of a while(1) loop in a script so as long as the crash is catastrophic, a new instance will be brought up immediately. Since all the clients automatically reconnect, the disruption will be equal to the time to restart the server plus the time for all the clients to reconnect. We measured server restart time as between 1.7 s and 2.7 s including the time to start up the Java virtual machine. Figure 36 shows the percentage of clients that have reconnected as a function of the time after the Event Heap server comes up.

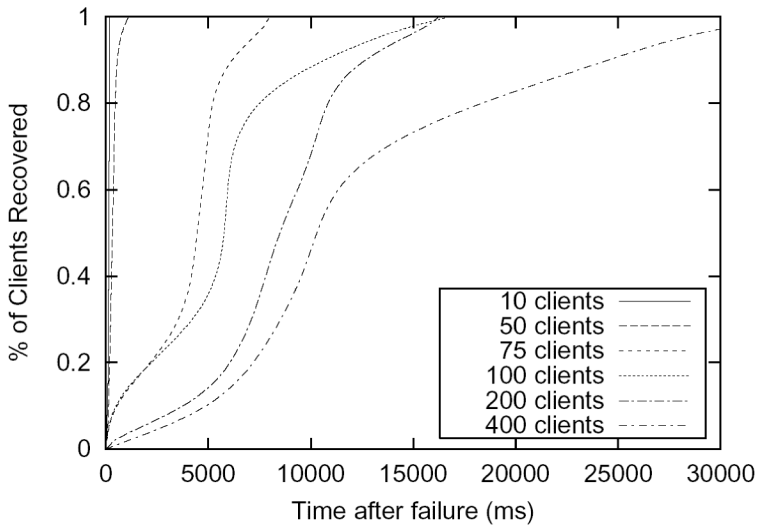


Figure 36 – Percent of Clients Reconnected vs. Time after Event Heap Server Crash

As the graph shows, for small numbers of clients, all of them reconnect within one second after the Event Heap server comes up. As a greater number of clients are trying to reconnect, however, it takes longer for all of them to become reconnected. At 200 clients, it takes about 15 s for all clients to reconnect. Including the time to restart the server that means an interruption of service in the interactive workspace of about 17 or 18 seconds when the Event Heap server crashes. This seems to be reasonable given that the Event Heap server was designed to be stable and should has few bugs that will lead to crashes.

The figure shows that reconnections come in waves. For example, the 75 client line shows that clients gradually connect out to about 3.5 s, and subsequently about 60% come in a 1s period. We suspect that this is being caused by the exponential back off in the underlying TCP/IP layer. All clients are waiting to reconnect, and an initial batch gets through. Those that don't get through time out and wait a few seconds before trying again. That batch then connects, and the process continues until all clients are reconnected.

6.3.3 Discussion of Performance Measurements

The experiments just described show that the Event Heap performance is more than adequate for our needs. One of the properties of the Event Heap model was scalability to workspace sized traffic loads (**P8**). In describing that property in Chapter 4, Section 4.1.8, we reasoned that the system needed to handle 100s of clients, and 10s to 100s of events per second all at a latency of

less than 100 ms for perceptual instantaneity (**P7**).[10] The graphs in Figure 34 and Figure 35 show that latency stays below 60 ms even out to around 300 clients and throughputs of 350 requests/s. In the deployed interactive workspaces we know about, there are on the order of tens of clients with request rates of tens of events per second, significantly lower than either our estimate for these figures or the maximum values the Event Heap implementation can handle. For these real-world values the latency is well below 20 ms, allowing up to five hops through the event heap while still maintaining perceptual instantaneity.

Compared to the Event Heap v1 implementation, the performance is much improved. Unloaded the latency of the system is less than 10 ms compared to 30 ms for the old version, and at 200 connected clients the latency is 20 ms versus between 100 ms and 2 s for the old implementation. In the cases being compared, the new implementation is between 3 and 100 times better than the old one. Although some of the difference may stem from the CPU speed difference (550 MHz for the old test vs. 800 MHz for the new test), this cannot account for a 3x improvement, much less 100x.

6.3.4 Performance Anecdotes

The measurements presented in this section show that the Event Heap implementation meets the traffic loads expected in an interactive workspace while maintaining perceptual instantaneity. It is also helpful, however, to look at real applications and see how they perform.

One anecdote in this respect is that initial testing has shown that the Event Heap can be used for sending mouse and keyboard events between machines with good perceptual smoothness. The PointRight mouse redirection system (see Section 6.1.1) Macintosh OS X version now uses the Event Heap. An informal test was done with several users controlling cursors on different remote machines from their laptops over a 802.11b wireless network. For casual use, the delay for controlling the remote machine was almost imperceptible, and even with several users moving their mice as quickly as possible the lag was tolerable. The motor-visual feedback loop between muscle-movement to control the mouse and visual perception of the cursor movement is known to be very latency intolerant, so even though this was not a formal test, the initial results are encouraging.

[10] 100 ms is to provide perceptual instantaneity of triggered actions. Cursor movement and other tightly coupled motor-visual interactions were not intended to be supported by the Event Heap and can require latencies of 10 ms or less.

The second anecdote is found in the iClub application programmed by a group of Stanford University seniors. This application plays music in the iRoom and synchronizes visuals running on the various displays to the music. It sends out music beats and frequency information at rates up to 170 times per minute (the beat rate of some the techno songs that are played). Despite this fairly high frequency, the visuals appear synchronized with the music.

THIS PAGE LEFT INTENTIONALLY BLANK.

Chapter 7 – Discussion

This chapter presents some discussion on various aspects of the Event Heap model. Among other things, it looks at potential weaknesses of the model, and at its suitability for coordination beyond interactive workspaces.

7.1 Comparing Event Heap Design to Internet Design

We were inspired in part by the designers of TCP/IP, the enabling protocol for the Internet, in choosing which course to take in investigating the domain of interactive workspaces. In a paper about the design choices taken when creating TCP/IP [45], Clark describes of the main decisions they made. Most of these correspond to choices we made in designing the Event Heap. While we don't expect the Event Heap to approach the success of TCP/IP and the Internet, we hope that following a similar design path bodes well for its future.

One choice made by the designers of TCP/IP was to allow it to work over any existing or future network. One alternative they could have pursued instead was to design a single media-level protocol that would run over a variety of transmission media (coax, twisted-pair, fiber, wireless, etc.). They comment that "while this might have permitted a higher degree of integration, and thus better performance, it was felt that it was necessary to incorporate the then existing network architectures if Internet was to be useful in a practical sense." Instead, they decided that networks supporting the transport of IP protocol packets could use any media level protocol and only needed to provide the ability to encapsulate IP packets and mechanisms for routing the packets on and off the network. Our decision to support legacy applications and devices was made for similar reasons: to be of any practical use the existing software and hardware that is in use by companies needs to be supported in their interactive workspaces. So, to support the Event Heap,

an interactive workspace need only support socket based communication, and for any given platform only a simple client API needs to be ported.

Another important goal for TCP/IP was to sacrifice some performance for better failure handling. Specifically, TCP/IP is set up so that clients were not signaled when synchronization was lost, but only when total failure occurred. This masked any transient failures in the intervening networks. We similarly chose to sacrifice performance for decoupled operation of applications and flexible means of routing events among the applications in an interactive workspace.

One decision that the designers of TCP/IP faced was whether to have a single protocol, or divide the standard into two pieces. They chose to separate the reliable transport protocol, TCP, from the basic packet forwarding protocol IP. In this way, not all applications need incur the performance penalty necessary for guaranteed reliable transport. This turns out to have been a very good choice since much of the audio and video that is sent over the Internet today tolerates loss better than high latency.

Unlike with referential-decoupling, which we decided was critical and worth some sacrifice in performance, we decided not to provide for transactions, persistence across crashes, total ordering or other potentially useful but performance sapping mechanisms at the level of the Event Heap. We reasoned that it would be possible to provide those mechanisms over the Event Heap by creating a higher level API if necessary. Further, we reasoned that many applications would not want to incur the latency overhead that some of those mechanisms entail. So far, this decision has proven to be the correct one, since the HCI researchers using our system want the latency of the system minimized for fluidity of interaction and have had no need for the described mechanisms.

At the cost of a potentially slower design, the creators of TCP/IP also chose simplicity wherever possible to minimize the difficulty of implementing the system on top of various other network infrastructures. Likewise, with the Event Heap and indeed all of the iROS system, we have chosen simplicity, adding the minimal set of features to provide for the characteristics of interactive workspaces and the needs of application developers in that space. On account of this, we have a compact implementation, at the expense of a rich feature set and performance optimizations for special cases.

7.2 Importance of Loose Coupling

Loosely coupled message-passing preserves existing fault isolation boundaries, which is particularly important in a dynamic environment such as an interactive workspace. In other

words, two applications that don't depend on one another but are able to modify their function based on events emitted by the other application will still have failure behavior that is independent of one another when coordinating through the Event Heap. This feature lets us quickly experiment with new devices and software without concern that we will destabilize the environment. We believe the *combination* of failure tolerance and interoperability we have been able to realize would not have been practical using any other mechanisms.

In fact, the predecessor to the Event Heap called netobj2 had these problems. It was a message passing system that created explicit socket connections to all participating applications and was fully decentralized. Applications had to be running at the same time and know the identity of other applications with which they wanted to communicate. It was fast, but not extensible (adding new devices or services was ad hoc) and not robust (the components themselves were buggy, and the system was not resilient to component failures, frequently wedging or requiring manual intervention). We cannot prove that loose coupling has made the only difference, but given our experience with iROS, the cost of indirect communications seems worth the benefit.

7.3 Potential Weaknesses of the Event Heap Model

Since the Event Heap was designed for the interactive workspace domain, it has some drawbacks that make it less useful outside of the domain. Due to the use of a shared medium, it is difficult to scale the system to large numbers of simultaneously communicating entities. Among other things, this makes it unsuitable for Internet scale coordination across tens or hundreds of thousands of devices. Scalability is less of an issue in an interactive workspace, however, where traffic load is bounded by the number of people that can meaningfully interact with one another and a set of devices to solve some problem.

While the indirect communication mechanism in the Event Heap model provides nice properties for the interactive workspaces domain, it forces two hops of communication for any event transmission between two parties (sender to server and server to receiver). This extra hop adds latency (although in [37] it has been shown that for a static communication pattern a properly implemented tuplespace can be made to adapt over time to single-hop communication). Both latency and scalability in an interactive workspace are, however, bounded by social constraints and human factors that make these drawbacks less of an issue. Specifically, latency on a local sub-net is small enough that even doubling it keeps the response fast enough to be perceptually instantaneous for humans.

Another potential problem is that all Event Heap state is centralized on one server machine. This makes the Event Heap server a single point of failure for all activity within the interactive workspace. To ameliorate this problem, we added the modular restartability feature (**F14**) so that Event Heap client applications don't need to be restarted if the server goes down. Since there are always humans in the workspace when applications are being used, one of them can simply restart the server (either the machine or just the application) when it goes down to restore the system to full functionality. Since clients buffer events between the time of an Event Heap server crash and the time when they connect to the new server instance, most events will be delivered using the modular restartability technique. It is still possible, however, that events that were received but not fully processed before the server crash will be lost.

To address the problem of lost events and to improve the reliability of the current implementation, there are several solutions that could be implemented to deal with server failure more automatically (although we have yet to implement any of them). The simplest method is to use hot-fail-over by running a back-up Event Heap server on a different machine that will take over should the primary server fail. Since the full server code is 100 KB and has a memory footprint of under 100MB, most modern machines could serve as a reasonable backup server. Further, since any critical state is beaconed, no checkpointing or other methods would need to be implemented to keep the state of the primary Even Heap server and the backup synchronized.

A more ambitious approach to dealing with server failure would be to write a fault-tolerant, distributed version of the Event Heap and run it across a small cluster of machines in an interactive workspace (thereby replacing the single-server with a small cluster). While requiring a greater coding effort, a distributed implementation would have the advantage of providing server performance that could be scaled with the number of machines dedicated to the cluster. Note that we don't believe a full peer-to-peer clusterized Event Heap implementation running across all the clients in an interactive workspace is a good idea. This would increase the complexity of each client making the client code less portable. Further, it makes clients more interdependent increasing the chance of failure-cascades. Beyond these arguments, we believe that the simplest implemented system that satisfies a set of design constraints is the most desirable one. A fully-clusterized implementation would add complexity without significantly improving the systems ability to provide for the failure tolerance design property (**P9**).

7.4 Applicability of Model

As discussed, this dissertation only argues for the suitability of the Event Heap model for coordination of user-controlled applications in an interactive workspace. Nonetheless, we believe that the model may also be useful in other sub-domains of ubiquitous computing. Specifically, we believe it applies for coordination of applications within any physically bounded region of human and device interaction. It could be used for coordination of devices people carry with them that are connected by a personal area network (PAN), devices in a home, or to allow interaction between built in devices and portable devices in public spaces—in restaurants, or on public transportation, for example.

As mentioned in the previous section, many of the features of the model come at the expense of scalability and performance. This limits the total number of devices and applications that could simultaneously coordinate using the system, prohibiting it from being useful in situations where thousands or tens of thousands of applications need to simultaneously coordinate with one another.

While tuplespaces were designed for use in high performance parallel computing applications, the extensions we have proposed to allow the interaction of a diverse collection of applications sacrifice much of the performance of the model. For example, self-description of fields consumes extra network bandwidth, and the more sophisticated matching semantics increase processor overhead. Therefore, our new model would be inappropriate for high performance parallel computing applications even within an interactive workspace.

Even within an interactive workspace, there are some types of communication among devices and applications that are not appropriate for an extended tuplespace model. In particular, it is not suited to streaming of high bandwidth data, such as sound or video, where throughput needs are beyond what the content-based matching system could reasonably handle. It is also not designed to support human computer interaction that requires tight motor-visual feedback, such as gesture interaction. In particular for gestures, several trips through the Event Heap might be needed to build up the gesture: low level movements need to be sent, recognized as a gesture, and the gesture needs to then be mapped to the appropriate application level action. Even with short transit times per pass through the Event Heap, the total latency could reach the point of being noticeable to users.

Another potential issue is the likely increase of the number of devices in an interactive workspace as hardware become cheaper and more plentiful. This leads to the conjecture that as time goes on

the centralized implementation will have to handle increasing traffic loads, and might eventually reach a breakdown point. While we can't be certain, we suspect that the processor power of the central server will scale at least as quickly as the number of devices, such that a centralized implementation will remain sufficient for a single interactive workspace. Specifically, processor power scales at an exponential rate according to Moore's Law, while we expect the number of devices in an interactive workspace to scale sub-exponentially, perhaps at a linear rate.

Chapter 8 – Related Work

This chapter presents research that is related to the work described in this dissertation. Unlike Chapter 2, which presented background information necessary to understand the contributions of this dissertation, this chapter relates efforts within ubiquitous computing in general, the specific sub-domain of ubiquitous computing rooms, and at developing other coordination systems (general categories of coordination were presented in Chapter 4, Section 4.5). Each piece of related work is compared to what has been presented in this dissertation.

8.1 General Challenges for Ubiquitous Computing

Researchers at PARC, the Palo Alto Research Center (formerly Xerox PARC) have been considering the challenges of deploying ubiquitous computing technology in homes of the future (Smart Homes). They report in [56] on seven challenges in deploying Smart Homes, many of which can also be applied to ubiquitous computing infrastructure for interactive workspaces.

The challenges are:

1. **The "Accidentally" Smart Home:** Technology will be brought into homes piece-meal. How will occupants deal with the unpredictable interaction of components? What are the boundaries of a smart home? How will occupants control the devices and the smart home as a whole? A similar challenge applies to non-technologically-enhanced conference rooms that gradually become interactive workspaces as technology becomes deployed in them.
2. **Impromptu Interoperability:** They say that "the chief obstacle limiting such impromptu interoperability now is that, in general, every device or software service must be explicitly written to understand *every other type* of device or software that it may

encounter." They further "believe that this challenge requires radical new models of connectivity and interoperability that reach beyond simple prior agreement on standard protocols and interfaces." This is a challenge that the Event Heap attempts to address in the domain of interactive workspaces by providing a centralized interchange mechanism that allows for adaptation of devices and applications to work with one another.

3. **No Systems Administrator:** Users can't be expected to be systems administrators. Two possible solutions to this are the "appliance model" where a repair-man is called in to fix things when there is a problem, and the "utility model," where most of the capability is in the network and not in the house, and thus can be fixed off-site. For interactive workspaces, we expect that some at least part-time system administrator will be available, so this challenge is less of an issue.

4. **Designing for Domestic Use:** Technology must be portable and flexible to occupant's requirements. One of the characteristics of interactive workspaces (**H2**) that the Event Heap tries to address is the same need to be flexible to accommodate users requirements.

5. **Social Implications of Aware Home Technologies:** Technologists need to be aware of social implications of technology. For example, how will privacy be effected? Will technology be labor saving, or just change the nature of labor? The challenge exists also for interactive workspaces—the technology deployed should actually make meetings and collaborative sessions more efficient, not just change the nature of how the time is spent.

6. **Reliability:** Systems can't crash all the time in a home. When individual devices break, the rest of the system can't go down. By being failure tolerant, the Event Heap strives to address a similar need in interactive workspaces.

7. **Inference in the Presence of Ambiguity:** Smart Home systems need to be very careful to do the right thing when acting on behalf of an occupant. When data influencing an action is ambiguous, the action should either not be taken, or should be easy to undo. If 'Clippy' in MS-Office can't get it right most of the time, it is unlikely in the more complex domain of a smart home that artificial intelligence (AI) will be able to do the right thing. According to the PARC researchers, "Systems that require omniscient understanding of human intent in order to function well, are perhaps better abandoned." Actions taken by a system on the users behalf need to be predictable, intelligible, and recoverable. This concern is one reason the Interactive Workspaces project has not looked into the use of AI in rooms.

The PARC researchers do present supporting evidence for each of their challenges, but do not report on any implemented software or system that they have built to address these challenges.

8.2 Ubiquitous Computing Research Projects

This section discusses research projects in ubiquitous computing that are related to, or have inspired, the work in the Interactive Workspaces group, and by extension, aspects of the Event Heap design.

8.2.1 General

We first present significant projects in ubiquitous computing that don't specifically address room environments, but nonetheless have some overlap with the work presented in this dissertation.

The Pebbles System

The Pebbles PDA Project [113] at Carnegie-Mellon University in Pittsburgh is investigating the use of collections of PDAs in collaboration with desktop PCs. One of the main goals of the project is to investigate what they call MMUIs, or Multi-Machine User Interfaces. Given the nature of the project, they specifically mean user interfaces that span handhelds and PCs. Three main issues that they are looking at with respect to MMUIs are: using multiple PDAs to simultaneously control applications on PCs, sharing information between the PCs and PDAs that are involved in the users' collaborations, and using handhelds as personal universal controllers.

In [113] they list nine applications they have implemented so far that demonstrate MMUIs. Listed below are some of the ones that are similar to applications developed as part of the Interactive Workspaces Project:

- **SlideShow Commander:** This is a companion application to PowerPoint, which runs on a PDA connected to the PC running the main presentation (preferably connected wirelessly). The PDA allows slide changes, seeks, annotations of the current slide from a miniature version on the PDA, and timing of the presentation. This application is similar to our SmartPresenter application (discussed in Chapter 6, Section 6.1.1). The main difference is that they use a one-to-one connection between a PDA and a PC due to restrictions in their coordination infrastructure, and therefore can't handle the multi-screen configurations found in interactive workspaces.
- **Military Command Post:** Public information is displayed on large screens in the rooms, and people can move information to and from their PDAs to the large screens. This is somewhat similar to multibrowsing (also discussed in Chapter 6, Section 6.1.1), except

that multibrowsing supports dynamic determination of targets through beacons placed on the Event Heap.

- **Remote Commander:** Allows multiple people to control the same PC from their PDAs. There are three modes: rotating control where at most one PDA can control the system at a time, overlay mode where multiple PDAs can simultaneously draw and annotate on an overlay layer without actually interacting with the running applications, and multi-user mode where all PDAs are simultaneously active for custom-applications that know how to handle multi-pointer interaction. This functionality is similar to PointRight (Chapter 6, Section 6.1.1), except that PointRight allows control of multiple target machines treated as a large virtual desktop. In contrast, Remote Commander supports multiple-cursors, but only on a single machine. The Event Heap provides PointRight with the ability to track changes in the machine configuration, a capability which is not needed for Remote Commander.

The Pebbles architecture is characterized primarily by a server running on the host PC. PDAs connect to that server using network sockets to control applications on the PC. On the PC side, plug-ins that allow control of applications are created for the Pebbles server and then become accessible to the PDAs using that server. Plug-ins may be specific to allow one application to be controlled, or general in allowing mouse-movement and keystrokes to be injected into the Windows task queue.

In comparison to the Interactive Workspaces project in general, the Pebbles project is more focused on interactions between one or more handheld device and a single PC, often called peer-to-peer interaction. Interactive Workspaces is looking at many-to-many interactions among groups of PCs, handhelds, laptops and other devices. By extension, the Event Heap model is more suited to this many-to-many environment, providing a central event store, which applications can use to anonymously exchange information. While the peer-to-peer nature of the Pebbles architecture restricts its utility in many-to-many scenarios, it does have the advantage that there is no central infrastructure to fail. As long as the user's PDA and the device or PC with which they are interacting are both up and running that local interaction can proceed.

HP CoolTown

The CoolTown project at Hewlett-Packard [94] is attempting to leverage the World Wide Web in the world of ubiquitous computing. They start with the assumption that most devices will be able to browse the web. Given that, their approach is to give a "home page" to every location to allow

users to interact with devices in that location, and to provide a web interface for every electronic device to allow interaction with and control of that device. To interact with the world around them, users can use their cell-phone, PDA or laptop to browse to the web pages for locations or devices. Since URLs are not generally known ahead of time, rooms and devices use local IR beacons to broadcast the URL for themselves. This beacon can be picked up by the user's device and used to look up and display the appropriate web page.

They also support data movement using web mechanisms and a technique they call eSquirt. Using eSquirt, a user could move a document to a printer for printing in the following way: the user selects the document and the printer, the document is moved to some web location, the URL of the document is beamed to the printer (this is the eSquirt), and then the printer retrieves the document from the URL location and prints it out.

The focus of CoolTown is on leveraging and augmenting the existing web infrastructure for ubiquitous computing rather than creating a new infrastructure from the ground up. The advantage of this is that they get access to all web content and the ability to use a relatively mature technology. The disadvantage is that they can't support legacy applications very well, nor can they support multi-device interaction. By comparison, both legacy applications and multi-device interaction are supported by design in the Event Heap.

Classroom 2000 Project (now eClass)

Researchers at Georgia Tech have been investigating the use of ubiquitous computing technologies in classrooms to facilitate learning since the mid-1990s. The project was initially called "Classroom 2000," but they have since changed the name to the more future-proof "eClass." The project is described in [16] and [17].

Their approach was to consider a classroom as a multimedia authoring environment, where the content presented by the professor, the notes generated by the professor and the students, along with the captured audio and video of what happened during the class form a learning artifact that students can return to after the class to review the material presented. To test their approach they ran several classes using large displays in the front of the classroom, some of which were touch-screens, where professors could present and annotate material. Students were given tablet computers or PDAs, which they could use to annotate copies of the material being presented in class. Notes taken by students and the professor were time stamped, and, after the class, a multimedia presentation of audio, video and notes could be browsed over the web. They did a study of students in one of the classes that used the system, and results were overall favorable,

somewhere between a three and four on a five point scale measuring ease of use, effectiveness and desirability of the technology (five being the best). The one thing that students did not like as well were the tablet computers and PDAs. They were found to be not as good as normal paper for note taking.

One thing that the researchers found to be very important was taking their technology "live." They say that: "The best way to understand the effect of ubiquitous technology in our everyday lives is to experience it." They found that trying to make a go of using technology, even if their software wasn't perfect, was very helpful. It helped them to see what was hard and to get feedback from real users. One of their findings was that simply keeping all of the equipment up and running was the hardest part. Compared to that, they found developing the tools to support the process relatively straightforward. We have also tried to follow this principle of using and deploying our technology in the interactive workspaces group, and agree with them that it is worth the effort.

Compared to interactive workspaces, the Classroom 2000 project focuses only on classrooms. The interactive workspaces project has also been focusing on team project-style rooms, and not auditoriums or lecture halls that might be used for classes. Classroom 2000 did not focus on any specific infrastructure, but rather wrote applications using whatever technologies were available—we believe that this may have contributed to their difficulty in keeping everything running. As such, they have no specific infrastructure that they developed against which the Event Heap itself could be compared. We suspect that creating and deploying their classroom applications would have been easier using the Event Heap.

The Context Toolkit

The Context Toolkit [54], like Classroom 2000/eClass, is a project at Georgia Tech. It is a toolkit and infrastructure system for allowing applications to use context information, such as location of an object or person, without having to include specific code to deal with sensors, data fusion and aggregation, etc. The toolkit provides a set of object types, each of which is run as a separate process on a machine in the context space. A context space is defined as the set of all machines running one of these objects that register with the same discovery service. The objects include widgets, which provide some piece of state (possibly from a sensor), services, which allow context state to be set, aggregators, which gather together context state from widgets, interpreters, which transform one type of context information into another, and the previously mentioned discovery service. Applications choose how the objects should be connected in an object graph, and then use resulting information to make decisions and set context state themselves via

services. Since the detailed context gathering is done in the objects, their code can be re-used between applications. Further, running objects can be shared among different applications.

Communications take place using XML messages sent between the objects using the HTTP protocol. The messages can request current context state, or specify a subscription to some specific piece of state that is tracked by that object. When state changes on an object, a message is sent to all subscribers with the update. There is no ordering of messages in the system across objects.

Objects register with the discovery service when they become available, and the discovery service pings objects periodically to make sure they are still alive. Currently the discovery service is centralized, and there is a scaling issue since it is not specified how you figure out what local discovery service to use.

Individual objects maintain state of the other objects that they are using, but will automatically delete the state if after a few attempts the remote object ceases to respond.

The Context Toolkit is written in Java, but since it uses standard wire protocols objects can be written in other languages. They have used C++, Frontier, Visual Basic and Python. The toolkit version they describe in [54] is about 12,400 lines of code. Their success in supporting other platforms suggests that an implementation of this size can be made to support a variety of languages. This corresponds with our own experience with the Event Heap implementation, which contains 11,286 lines of code.

The Context Toolkit addresses a different need from the Event Heap. The Event Heap mechanism provides a means of coordination among applications and devices in a space, whereas the Context Toolkit provides a means for applications and devices to find out about their environment. The Context Toolkit could be built on top of the Event Heap, but would need to add support for interpretation, and aggregation. The Context Toolkit could also potentially be used as a location service that would allow devices to automatically select the correct local Event Heap to use.

SOLAR

The SOLAR system [40] is work done at Dartmouth. Like the Context Toolkit, the goal of SOLAR is to provide a means for applications to access information about their local environment. It provides a mechanism for aggregation and distribution of event streams, presumed to be originating in sensors or applications reporting on the state of the environment. It allows streams to be linked through filters, aggregators, etc. before arriving at one or more sink

applications. One issue with the system is that any path of transformations is named explicitly by the set of objects that touch the events as they flow through the transform graph. This hard coding makes the system more rigid, and potentially more failure prone. To distribute final aggregated results to applications a publish-subscribe system is used.

Like the Context Toolkit, SOLAR addresses a different need than the Event Heap. The publish-subscribe system that it uses for distribution could be replaced with the Event Heap allowing for the coordinating applications in an interactive workspace to have access to context information.

8.2.2 Ubiquitous Computing Rooms

Here we describe other projects looking at ubiquitous computing rooms. We begin with efforts that are more recent or ones that are more similar to the work in the Interactive Workspaces group.

The i-Land Project

The i-Land project in the Ambiente Group at GMD-IPSI in Darmstadt is probably the project most similar to our own. Their prototype environment is called the Ambiente Lab (after their research group), and their layout was part of the inspiration for the configuration we chose to deploy in the iRoom. The i-Land project as a whole is described in [138] and [136].

The stated goal of their project is to create integrated physical and virtual workspaces. Part of this investigation has led them to develop three types of "electronic" furniture, called Roomware [138]. They have constructed prototypes of the interacTable (a table top display), commChairs (special chairs with built in displays or laptop connections), and the DynaWall (a three-panel large electronic white-board display composed of three SmartBOARDs which is similar to our iRoom setup).

They have also developed a system, called *passage,* that allows association of digital objects with physical objects for movement around the room. This is done by associating a physical object's weight with a reference to the virtual object, and then calling up the virtual object as the physical object is placed on scales around the workspace. Passage is their mechanism for what we identified as the need to move data in an interactive workspace (which was described in Chapter 3, Section 3.4).

The i-Land project has focused on fully custom environments for ubiquitous computing rooms and the investigation of Human Computer Interaction (HCI) techniques that are needed in these environments. For example, while in the iRoom each computer is stand-alone and we provide

techniques for moving data among machines and remotely controlling machines, they run custom software that allows all of the screens in the space to be part of one large display environment. Windows can be moved across display boundaries without difficulty. Within this space, objects are shared so two users can access the same sketch and simultaneously modify it on different displays.

The focus of the i-Land project on sharing objects and simultaneous interaction has led the researchers there to develop the BEACH infrastructure. It is described in most detail in [139]. Like the other software created by the i-Land project, it is written in SmallTalk and only supports that language. Adapting the framework to support other platforms would probably be difficult as their implementation relies on many SmallTalk specific features.

The BEACH system contains two main pieces: a conceptual model for synchronous applications in ubiquitous computing environments, and a concrete software infrastructure to support the BEACH model. This is superficially similar to the division between the Event Heap model and the Event Heap implementation in this dissertation, but their model actually describes how to structure new applications for ubiquitous computing rooms rather than describing the necessary infrastructure properties and features for their implementation.

Model Class	Description
Data Model	Objects that contain data state and allow modification of that state.
Application Model	Objects that contain methods for interacting with data and store the current state of that interaction.
User Interface Model	Objects that represent user interface elements and techniques.
Environment Model	Objects that reflect the current state of the environment.
Interaction Model	Objects that allow for interaction with users on specific devices (mouse input, display output)

Table 15 - Model Classes in the BEACH Conceptual Model

They derived the BEACH model by first determining the various requirements that stem from HCI, ubiquitous computing, CSCW and systems issues that constrain any application model that would be constructed for this space. There are three dimensions of their conceptual model: separation of concerns, coupling and sharing, and level of abstraction. The first dimension is concerned with dividing up potential software components into their purpose in an application for a ubiquitous computing room. The second dimension provides for specification of whether a component will be shared across multiple machines or resides only locally on specific devices. The third and final dimension determines the level of abstraction at which a specified component is intended to be used.

Within the first dimension, components are divided up into the following types: data model, application model, user interface model, environment model, and interaction model, with all but the final type falling into the category of 'shared' on the second dimension. Data model objects are used to store and manipulate actual data. Application model objects contain methods of manipulating data objects and store the state of interaction with a data object (e.g. line 3, char 5 is the current cursor position). User interface model objects specify how output controls look and how they can be manipulated at an abstract level (e.g. a scroll-bar widget which is independent of type of screen or type of pointer input). Environment model objects represent physical-objects and their state in the real world (e.g. topology of screens in a space). Finally, interaction model objects control the physical input and output systems of specific devices (e.g. how to output to a screen or read inputs from a mouse).

Abstraction Level	Description
Task Level	Application and task specific objects
Generic Level	Reusable components suitable for use in some general class of applications.
Model Level	Non end-user accessible objects usable in some given model class.
Core Level	Low level or cross-model-class basic functionality objects.

Table 16 - Abstraction Levels in the BEACH Conceptual Model

The third dimension contains four different levels of abstraction that can be used to categorize components of any given model class. The first level of abstraction is 'Core' and consists of components that deal with devices or object interaction at a low level (e.g. the shared object space libraries that are part of the infrastructure). Object classes in this level may be applicable to multiple different models in the first dimension. The second level of abstraction is 'Model', and objects in this category provide for reusable functionality within a single model class; they are helpers for developers only, however, and never contain functionality specifically for end users. The next abstraction level is 'Generic' and represents objects and classes that apply to some general category of applications and are reusable across applications for some specific model class, such as text editing; they may expose behavior to end-users. The final abstraction class is 'Task' level which represents objects and classes that are specific to the implementation of some task or application.

Their contention is that applications in ubiquitous computing environments can all be represented as the interaction between modules or objects, each of which can be described by its model class, sharing/coupling degree, and level of abstraction. Further, when designing applications, if authors insure that specific objects and classes don't span multiple levels or model classes, they assert that implementation will be clean and reusability of the components will be maximized.

They also have a graphical technique to represent how any given application for a ubiquitous computing room is decomposed within their conceptual model.

A major difference between the i-Land work and the work we have done as part of the Interactive Workspaces project is that we don't have any equivalent to the BEACH conceptual model. We provide a general set of tools with appropriate properties and allow application developers determine for themselves how to use them. This is a potential drawback for our system. We believe that the BEACH conceptual model could be used to structure applications and ensembles for implementation using the Event Heap and the other iROS system tools.

The implementation of the BEACH model consists of a set of basic modules to help users construct applications in a ubiquitous computing room. The modules they provide all can be categorized themselves within the BEACH conceptual model.

The module that is most responsible for coordination within their ubiquitous computing environment is their shared object system, which they categorize at the core level spanning all the different model classes. The sharing is provided by a system called COAST that was originally designed for different-place/same-time CSCW applications and has been adapted in BEACH for ubiquitous computing room environments.

COAST itself has been described in [130] and [131]. Shared objects are stored both on a central server and as copies on every participating machine. A sophisticated locking and resolution mechanism is provided so that simultaneous object modification is permitted as long as the underlying data being modified is orthogonal. Changes are committed optimistically in the sense that a local commit is performed before confirmation is received that the change has been accepted by the server. This allows quick feedback in local UIs at the cost of a mandatory rollback should the change be rejected. The system also contains a session concept to allow the grouping of shared objects. This allows the system to tailor performance by restricting coordination needs to sub-sets of all of the objects. One big advantage of COAST is that it allows developers to create synchronous access applications while using objects that behave similarly to local objects. Two acknowledged drawbacks are that they don't support disconnected work, so the system doesn't work well in situations with dynamic change (**T3**), and since objects are replicated, there is some startup overhead to initially get data copied to participating machines.

Compared to the Event Heap, the COAST framework is more tightly coupled since applications must maintain connectivity to insure that the shared state remains valid. In other ways it resembles a publish-subscribe system since applications effectively subscribe to be notified when

updates to the shared objects take place. Unlike a publish-subscribe system, communication is opaque, making it potentially more difficult to debug and extend applications or application ensembles running in their system. Developers are also restricted to using SmallTalk. Their goal, however, was to design tightly integrated applications that run across devices in their space, and for this type of application their infrastructure provides much more assistance to the developer than the iROS system. We suspect that by adding some mechanism similar to COAST, the iROS system would be better able to support cross-machine, synchronous, applications for those situations where this is desirable.

Gaia OS

The Gaia project [39] is also investigating ubiquitous computing rooms. While the previously discussed i-Land project approaches the ubiquitous computing room domain from a primarily HCI perspective, the Gaia project comes from the operating systems perspective. As far as we understand, they don't have any prototype environment that they are using to test their system.

They have a model for applications in their system called MPACC (Model, Presentation, Adapter, Controller, Coordinator), which defines how applications can be designed for these environments. The model is an extended version of the model-view-controller approach (MVC). It goes beyond MVC by allowing for multiple interfaces to an application and by allowing adapters, which permit applications and interfaces to work together, which were originally not intended to do so. The MPACC model is in some ways similar to the BEACH conceptual model discussed earlier in this section, although it is not as comprehensive (it doesn't address environmental state access for applications, for instance).

The main Gaia initiative, however, is to make a meta-operating system for ubiquitous computing rooms that runs on top of existing devices and operating systems and exposes to developers a common abstraction of a physical space. This is a more monolithic approach than what we have been taking with iROS, but overall they provide many of the same facilities. They provide the actual OS, based on CORBA, which allows object exchange, a publish-subscribe type event notification system, discovery, and so on. They also have a simple scripting language called Lua, which allows Gaia components using CORBA, COM, and Java to be glued together.

The key underlying coordination mechanism for the Gaia OS is the Universal Object Bus (UOB). It provides the following four key abstractions:

- **Unified Component:** All components in the system conform to this standard, which allows them to be controlled and managed regardless of location.

- **Component Manager:** The component manager provides an interface to manage the life cycles and interactions of unified components.
- **Component Container:** Essentially a JVM in which components are run.
- **UOBHost:** Any device capable of running component containers and components.

Like us, their goal is to provide application portability, but we are seeking to provide a standard interconnection framework for applications rather than attempting to abstract away the diversity of devices and applications found in such spaces. We see their approach as promising for designing new applications to work in an interactive workspace, but it is unclear how well it will be able to handle legacy applications and the incremental evolution of a space with the addition of new devices.

MIT Intelligent Room

Part of Project Oxygen [11], an umbrella project to fund research into ubiquitous computing at MIT, the Intelligent Room project [46] is investigating intelligent environments that can anticipate and respond to the needs of occupants. They have created a ubiquitous computing room with speech recognition, multiple displays, and cameras to track users and detect gestures. The goal of their project is to provide unencumbered user access to the rooms technology by using vision techniques and speech recognition. Their focus on artificial intelligence differentiates them from the Interactive Workspaces project since we do not attempt to put any intelligence into the room, but instead attempt to provide the user with intuitive mechanisms for controlling the environment.

Their basic approach is to use many different AI techniques (vision, speech recognition) running on different PCs, and then bind them together using an agent infrastructure into something they call the *scatterbrain*. The system as a whole then tries to infer what users are trying to do. For example, if a user points to a screen and says, "computer click this link," the voice recognition system decides something needs to be clicked, then retrieves from the vision system the name of the object to which he is pointing. While it may seem that this is just a matter of bolting together systems from elsewhere in AI, they say the main challenge is in maintaining a global sense of state and activity derived from the various sub-systems in use. A main claim of their work is that they are trying "to bring computation into the realm of ordinary, everyday activity in a seamless way."

The agent infrastructure they use to coordinate the various components and AI routines in the space is called the Metaglue System [47]. The system is written in Java and, among other things,

provides a basic class that can be extended to make something an agent. Agents specify any hardware dependencies they have, and also what other agent types they rely upon to accomplish their task. All agents run inside a modified JVM called the Metaglue Virtual Machine (MVM), of which one is run per machine. To start a desired agent, the user just runs the agent on any MVM in the room, and the system will then start it on a machine somewhere in the environment that satisfies the agents hardware requirements. The system will also connect the newly starting agent to existing agents it needs to access, and start any other necessary agents and connect them as well. Those agents that were triggered to start may in turn cause additional agents to be started until the whole system is up.

The Metaglue system provides additional support for agents to store and retrieve state, and to stop running agents and restart them for debugging reasons without the whole system going down. Finally, there is a facility for agents to broadcast when interesting events happen, and other agents to subscribe, allowing for some basic publish-subscribe interactions.

The Metaglue system was designed to support the following needs they found to be characteristic of intelligent environments:

- Interconnect and manage large numbers of disparate hardware and software components.
- Control assemblies of interacting software agents *en masse*.
- Operate in real-time.
- Dynamically add and subtract components to a running system without interrupting its operation.
- Change/upgrade components without taking down the system.
- Control allocation of resources.
- Provide a means to capture persistent state information.

Unlike our work with the Event Heap, little emphasis is placed on failure tolerance, legacy applications and debugging in the Metaglue system. Their system is of the RMI-Rendezvous type described in Chapter 4, Section 4.5.1 and has the same drawbacks discussed there. For the reasons discussed there, we don't believe it to be a very good general-purpose coordination environment, although it seems well suited to the collections of agents they use in their work. To lessen the coupling inherent to RMI, calls in their system are, however, routed through an intermediary that forwards them to the appropriate final target. This allows the infrastructure to switch the bindings between objects on the fly.

Microsoft EasyLiving

Another project that is looking at how to build intelligence into ubiquitous computing rooms is the EasyLiving project at Microsoft Research [34]. Their focus differs from the Interactive Workspaces project, where we look at providing affordances to users to perform actions but do not investigate how to intelligently respond or act on behalf of the user. They have created two main components for their environment: a geometric location tracking system and a middleware infrastructure. Their project has two general goals:

- Aggregate diverse devices into a coherent user experience. This is similar to the high-level goal of the Interactive Workspaces project.
- Assume a set of mostly independent devices, not a room full of peripherals connected to a central machine.

The geometric tracking system abstracts away the means of tracking so that applications can focus on using the tracking information. In their current system, they use tracking to allow control of the most appropriate local display based on a user's position and orientation and the keyboard and mouse they are using.

The middleware is an asynchronous XML-based message passing system called InConcert. Objects themselves are named, so messages can be correctly routed even as the objects are moved among the various machines in the environment. They claim three major benefits to using InConcert:

- Makes it possible to develop components relatively quickly.
- Allows components to be conveniently moved between devices.
- Keeps the user experience responsive even when the device is isolated from all or part of the network due to the use of asynchronous communication.

Unfortunately, the InConcert system is not described in sufficient detail to make an accurate comparison with the Event Heap. The asynchronous messaging they use does provide for temporal decoupling (**P1**), but they appear to reference objects by name, so there is referential coupling (the opposite of referential decoupling, which is specified in property **P2**). On most of the other desired properties, it is not evident where their infrastructure stands.

Envisionment and Discovery Collaboratory (EDC)

Researchers at the University of Colorado at Boulder have been investigating a ubiquitous computing room they call the "Envisionment and Discovery Collaboratory" or EDC [23]. Their

environment is what might be termed an 'iRoom Lite' with a single SMART Board table and another vertically oriented SMART Board. The EDC was built to investigate how to create coevolutionary environments where the users change as they learn from the environment, and the environment adapts (or can be adapted) to the changing needs of the users. It was designed to explore team-based design exercises that are not based around the use of a traditional PC.

They use the table as an *action* space, and the other SMART Board as a *reflection* space. The former is used by a team to interact with a simulation to attempt to come up with a satisfactory design (in their example this is creating a bus route that will encourage fewer people to drive cars). Physical items can be placed onto the screen to represent placement of items in the simulation. The reflection space displays background information that might be useful in guiding the simulation running in the action space (in their example this included studies on public transit, etc.). The users may also post insights or observations that arise during their interaction with the simulation into the reflection space for future use.

They have no custom designed infrastructure to support their work, but instead use other programming toolkits not originally intended for this specific domain (specifically, they use AgentSheets [127], Visual AgenTalk [126], and DynaSites, an in house tool for making evolvable web-based information spaces). Although they have no custom infrastructure against which the Event Heap can be compared, it might be possible to replace the non-specific toolkits they are using with the Event Heap.

Reactive Environments

The University of Toronto did work on what they call "Reactive Environments" [48, 49] in the mid-1990s. They looked at room-based video conferencing systems with multiple seats and multiple displays. Their goal was to make a system that was more intuitive than then-current state of the art systems, and at the same time make the failure modes less frustrating. The overall system they arrived at after several iterations automatically switches video and presentation material to the main display as needed, and re-routes what users are seeing on their per seat personal video-conferencing displays as well. The system provides a manual override system to route video explicitly by touching LED buttons on the sources and destinations of video in source-destination order.

This project did not look at systems infrastructure but focused instead on the human factors in these environments. The work they did could be used instead to inform construction of similar video conferencing systems in interactive workspaces built on top of the Event Heap and the

iROS system (some work on video conferencing in the iRoom has already been done by Milton Chen, a member of the Interactive Workspaces group [41]).

A Room Management System

In [62], Falkowski looks at a "Room Management System." Their environment consisted of PCs on rollable carts and electronic whiteboard systems. Their room management system tracks the position of the physical PCs and whiteboards in the system and keeps information on who is currently using them. From a system control application running on a central machine, applications can be started or killed on the machines in the room. Web pages can also be loaded on some or all of the machines in the space from the central machine (somewhat like the multibrowsing system that is part of iROS, but not as flexible since there is no peer-to-peer capability for pushing and pulling web content). Additionally, the administrator using the central control system can see what is running on machines in the room, and cause windows to be minimized, maximized, closed, etc. Their implementation uses direct socket connections from the central machine to a special service running on each Windows PC, although both unicast and multicast are used to control the machines. Since the Event Heap supports both of these routing patterns, it could be used to recreate their room management system using it as the coordination mechanism.

Compared to work in the Interactive Workspaces project, the scope of this project was more limited. Their coordination mechanism is master-slave based, so it provides little flexibility. Since there is no peer-to-peer interaction, provided the controller machine is robust the system as a whole should be tolerant of failure in any one peer. Their interface with the slave PC windowing environments could potentially be added to iROS to allow better control of legacy operating systems.

The Colab

The Colab [134] was an interactive-workspace-like conference room built at Xerox PARC in the 1980s. It consisted of a large projected screen at the front of the room and individual workstations for each conference participant embedded in a U-shaped table that faced the screen. Users could work locally, or take control of the front screen. The system was for group brainstorming.

The basic function of the software allowed each user a private area for taking notes, and a public what-you-see-is-what-I-see (WYSIWIS) area that was shared. The front screen could display one users screen including their private area and the public area. Users could open separate small edit

windows to create new items, and then send them to the shared public area. Not much detail is given about the implementation, but it seems to have been custom implemented using sockets or some similar technique. Constructing a similar system using the Event Heap would be straightforward: a view application could be used and set to emit events indicating changes in the viewer, it could similarly pick up the same events to synchronize with other viewer applications in the interactive workspace.

From their paper [134], their were problems with the various incarnations of the software that they wrote they were only able to create a few versions. We believe that if they had had some framework similar to the Event Heap on which to build, they would have been able to prototype more versions to come up with a more optimal implementation..

8.2.3 Summary

Table 17 contains a summary of the similarities and differences between each of the just described ubiquitous computing research projects and the Interactive Workspaces project. Where applicable, comparisons to the Event Heap model are also given.

Research Project	Similarities	Differences
General		
Pebbles	• Focus on Multi-Machine User Interfaces (MMUIs)	• No multi-machine model (PC to PDA peer-to-peer interactions only)
HP CoolTown	• Binding virtual objects to real-world locations • Leveraging web infrastructure	• No general mechanism for inter-application coordination
Classroom 2000	• Philosophy of "trying things out" by deploying research software in real situations	• Focus on classroom situations • No systems infrastructure research
Context Toolkit	• Provides multi-platform support for an infrastructure of similar complexity to the Event Heap	• Focus on gathering and providing context information
SOLAR	• Uses a publish-subscribe mechanism, which provides some decoupling	• Focus on gathering and providing context information
Room Projects		
i-Land	• Their prototype interactive workspace inspired the iRoom setup	• Focus on synchronous multi-machine applications • Fully SmallTalk based system (no support for heterogeneous software)
Gaia OS	• Goal of providing Meta-Operating System for ubiquitous computing rooms • Goal to support heterogeneous environments	• More focus on abstracting away heterogeneity • Less focus on fault tolerance • No prototype environment for testing
MIT Intelligent Room	• Focus on interactive-workspace-like environments with lots of displays, devices, etc.	• Focus on making room "smart" (AI approach) • Infrastructure is more tightly coupled (RMI/Rendezvous based)
Microsoft EasyLiving	• Goal of supporting legacy applications and systems	• Focus on responsive environments and anticipating users' needs (AI approach) • No general coordination infrastructure
Envisionment and Discovery Collaboratory	• Research on interaction with multiple large screens in a workspace	• Research focus on techniques for user interaction with sets of information • No effort toward general systems infrastructures in their space
Room Management System	• Support for legacy applications	• Only client-server communication supported • No general coordination mechanisms
The Colab	• Investigation of private versus public and data movement between the two	• Custom implementation of specific applications only • No general infrastructure development

Table 17 –Research Project Comparison

8.3 Tuplespace-based Coordination Systems

The set of coordination infrastructures described in this section are all extended versions of the tuplespace model. Linda, the original tuplespace implementation, was described in the background material of Chapter 4, Section 4.2.1 and is therefore not described again here. As a whole, Chapter 4 details why tuplespaces are a good starting place for a coordination model in an interactive workspace and describes the extensions that were needed for the Event Heap model.

8.3.1 JavaSpaces/GigaSpaces

JavaSpaces [7] is a Java based tuplespace implementation that is distributed with Jini. Technically, JavaSpaces itself is a specification, and only a reference implementation comes with Jini. There is also a commercial implementation of the JavaSpaces specification called GigaSpaces [2], which has some additional features beyond the base standard including limited support for C++. Here we describe basic JavaSpaces since beyond performance differences the two are similar.

The main distinguishing feature of JavaSpaces is that tuples are Java objects. This allows all the standard object oriented features such as typing and using template objects to match sub-classes. The chief drawback of the system is that it requires both Java and Jini, and is thus more useful for creating distributed systems as a whole than it is to provide a glue to allow coordination across disparate platforms. JavaSpaces was included with Jini to provide for distributed persistence and to better support the design and implementation of distributed algorithms. Some of the other goals the designers had when creating JavaSpaces were:

- Client side should have few classes to make class download fast, and keep the API simple.
- Client side footprint should be small to run on impoverished devices. We had a similar goal for the Event Heap.
- A variety of implementations should be possible for the back-end.
- Should be possible to create a replicated service.

Some of the main features of JavaSpaces are:

- Support for a transaction mechanism with ACID properties for multi-operation or multi-tuplespace updates.

- Entries (tuples) are governed by a lease that allows them to expire. This is one of the needed extensions we found for the Event Heap to provide limited data persistence (**F4**).
- Not intended primarily as a data repository.
- JavaSpaces classes download and execute on the client. This allows different back-ends, since the downloaded code stubs can have their own unique protocols. This is what enables off the shelf client applications that use JavaSpaces to switch to commercial implementations like GigaSpaces without modification.
- Allows objects used repeatedly as templates to be serialized once, and then subsequently a snapshot version can be sent. The Event Heap reference client supports the same functionality.

Aside from the above, here are some of the other ways that JavaSpaces differ from basic Linda-style tuplespaces:

- Has strong object oriented typing of tuples.
- Tuples are objects, so they may have methods.
- Matching of sub-types is allowed.
- Fields are objects in and of themselves.
- No 'eval' to launch new threads. Most modern extended tuplespaces including the Event Heap have also eschewed this feature.

It would be possible to build the Event Heap as an extension of JavaSpaces, but its Java-centric nature would make it difficult to provide for clients on different platforms.

8.3.2 T Spaces

T Spaces [145], developed by IBM, is another Java based extended tuplespace implementation. It was originally derived from a fast, in-memory, database system, and one of its key features is the ability to perform queries which go beyond the standard Linda-style tuplespace template tuple-matching semantics. Some of the requirements that the developers had for the system were:

- Footprint small enough for even embedded devices. This was also a specified requirement for the Event Heap and JavaSpaces.
- Flexible enough to accept new data types.
- Provide asynchronous messaging service.

- Must support anonymous communication.
- Should have a data repository for simple data.

Some of the specific features that are provided by T Spaces that go beyond the features of a basic Linda-style tuplespace are:

- **Operator superset:** Includes scan, consuming scan, and Rhonda atomic tuple exchange operator. The first two are similar to the 'snoop' operator in the Event Heap, with consuming scan being destructive.
- **Dynamically modifiable behavior:** New operators can be uploaded and used in the server using Java byte code exchange.
- **Persistent data repository:** Allows transactions and option to back tuplespace with persistent store.
- **Database indexing and query capability:** Can use queries that are more powerful on tuples in the space and the server indexes tuples for faster search.
- **Access controls:** User and group permissions can be set for each tuplespace.
- **Event notification:** Applications can register for callbacks when events are posted to the server. This provides query persistence, which was one of the needed tuplespace extensions to provide for coordination in an interactive workspace (feature **F5**).

The system is implemented in a similar manner to the Event Heap, with a single server and clients that create a socket connection to the server. As with the Event Heap, a single TCP/IP connection from each client to the server is maintained. Calls are all non-blocking, so a request is sent out and then a reference to the caller is stored. When the server returns a request, the thread that made the original call is woken up and passed the result. The server launches a new thread to handle each incoming client request. In this area, the Event Heap implementation is slightly more efficient, maintaining a dispatch queue and a fixed pool of worker threads. This avoids the overhead of having to continually start new threads and let the old ones get garbage-collected.

The similarities of T Spaces and the Event Heap are not entirely coincidental since the original two versions of the Event Heap were built on top of T Spaces, using an unmodified T Spaces server as the Event Heap server. Many of the features of T Spaces therefore inspired the actual full Event Heap implementation. On the other hand, we found that we never used some of the T Spaces features such as transactions, the sophisticated database querying mechanisms, and the

ability to connect a persistent database to back the tuple store. Since all of these features added to library size and reduced performance, they were left out of the Event Heap model and implementation.

Other papers on T Spaces include [32] and [99].

8.3.3 LIME

LIME [111, 120, 121] stands for "Linda in a Mobile Environment." It is a tuplespace system that takes into account that as agents (processes) move around they will be moving in and out of contact with other agents, either due to physical or code mobility. This mobility is not well provided for in the standard tuplespace model, so LIME provides a per agent tuplespace, known as an interface tuple space (ITS), to which all tuple operations are submitted. When the agent is co-located on a host with other agents, the ITS's of all of the agents become mutually accessible to one another. When an agent leaves, the tuples in its ITS leave with it. In addition, a federated tuplespace can be created between hosts, in which case all ITS's across the hosts in the federated space are mutually accessible.

To provide for better routing of tuples, all tuples may be tagged with the name of the destination ITS. If the destination ITS is currently unreachable, the tuple will be held in the local ITS until such time as it may be successfully delivered to the proper ITS. This insures that tuples intended for a given agent will eventually reach it, and will remain associated with it even when it or other agents disengage from one another. When querying, it is possible to restrict the search to a specific ITS, and further to tuples with a certain target ITS within a given ITS.

While the ability to route tuples adds flexibility to the system, this comes at the cost of requiring applications to track the identities of each ITS (and by extension, the corresponding device). Given the highly mobile environments targeted by LIME, we believe the routing feature could lead to brittle applications if developers don't take care to account for the disappearance of devices and the corresponding possibility that tuples tagged for the ITS associated with the device may never be delivered. We are therefore uncertain whether this routing ability is more of a benefit or a detriment to the LIME system.

An addition to the system is what they refer to as 'reactive' programming, which essentially allows the system to have a publish-subscribe mode (providing the 'Query Persistence/Registration' feature, **F5**). In this mode an agent subscribes to have a method invoked upon the appearance of any tuple in the local ITS matching a specified template. This insures that all tuples can be seen before they are removed, which could happen if a tuple were

removed while the agent was processing. For technical reasons, the subscribe, which they call **reactsTo**, only works on the local ITS. To allow reactions to tuple placement in other ITSs in the federated tuplespace, they also provide an **upon** call. This works by registering a **reactsTo** call in all other connected ITSs, where the program to be executed is the placement of a version of the triggering tuple encapsulated with notification details into the ITS of the agent that registered the **upon** request. When the notification tuple appears, it is automatically retrieved and delivered to the method registered in the **upon** call.

The LIME system is Java only, although they do all communications via sockets, so, provided no Java specific information is being sent over the sockets, it might be possible to implement clients in other languages.

LIME differs from the Event Heap primarily in its focus on providing for ad hoc mobile interactions among devices. In such scenarios, they do not have the luxury of assuming a relatively slow changing environment, such as a room, where a permanent server can be located. On the other hand, since they have focused on the mobility issues, LIME lacks flexible-typing, self-describing tuples, and tuple expiration, features which we have found to be important in the interactive workspace domain.

8.3.4 L2imbo

L2imbo [31, 52] is a system that uses multiple extended tuplespaces to provide quality of service (QoS) channels for inter-application communication. The developers propose that mobile applications should do all communication through tuplespaces, with each tuplespace being used as a channel with different characteristics. Each tuplespace would have a different QoS, and applications could subscribe to the tuplespace whose QoS constraints most closely match those desired by the application, and for which the application has permission to join. Different QoS for a video stream, for example, would be supported by having different bandwidths being sent through different tuplespaces. Filter programs would down-sample the data from higher bandwidth tuplespaces and place it into the lower bandwidth tuplespaces.

A universal tuplespace would be used to list the available tuplespaces and their attributes. The universal space, or sub-spaces, could also be used as locations to which context and monitoring information could be posted. In particular, the paper advocates that the space and time decoupling offered by tuplespaces are useful to deal with the dynamic and heterogeneous environments encountered by mobile applications.

Some of the key extensions they have that go beyond the basic Linda-style tuplespace model are:

- Multiple tuplespaces with different QoS attributes.
- Explicit tuple type hierarchy with sub-typing.
- QoS management architecture supporting monitoring and adaptation.
- Expiration times and priorities for tuples. This provides the limited data persistence (**F4**) we found necessary for interactive workspaces, and is a feature also found in JavaSpaces (through leases) and in T Spaces

One interesting thing that they do with expiration is to allow an expiration to be specified for both posts and receives. In the case of receives, they take the expiration to mean the amount of time to wait for a match before returning control to the application. This fits nicely with the idea of the duality of data persistence and query persistence.

Their overall implementation uses distributed tuplespaces with scalable reliable multicast (SRM) used to distribute queries and posted tuples.

While using the filtering and interposition capabilities of tuplespaces to provide for different qualities of service seems reasonable at face value, based on our experience with the performance of tuplespace implementations we are skeptical that this is the right approach for distributing high bandwidth content like video.

8.3.5 Modern Tuplespace System Comparison

Like the Event Heap, all of the just described coordination systems are based on the tuplespaces. Each one also has some sub-set of the extensions to the basic tuplespace model which are necessary in an interactive workspaces environment (as presented in Chapter 4, Section 4.2.3). While between the four systems, every proposed extension is implemented (as well as many extensions we found unnecessary), no single system has all of the extensions found in the Event Heap.

Table 18 summarizes where each system falls with regards to the proposed extensions. Although not proposed as a desired feature since it lies at the implementation level, cross-platform support has been included in the table as many of the systems are fairly tightly tied to the class system of a particular language (most notably Java), which limits their ability to provide for the heterogeneity we found to be characteristic of interactive workspaces (**T1** and **T2**).

The four tuplespace-based systems that appear in the table are Gigaspaces [2], T Spaces [145], L2imbo [52] and LIME [111]. Gigaspaces was chosen instead of JavaSpaces because it is a commercial product with slightly more features than the reference implementation.

Extension	Gigaspaces	TSpaces	L2imbo	LIME[7]
Flexible Typing	~[1]	~[5]	✓	×
Self-describing Tuples	✓[2]	✓	×	×
Standard Routing Fields	×	×	×	~[8]
Tuple Expiration	✓[3]	✓	✓	×
Registration	✓	✓	×	✓
FIFO, At Most Once Ordering	×	×	×	✓[9]
Modular Restartability	×	×	×	✓[10]
Platform Independence	~[4]	×	✓[6]	×[11]

1. Standard Java Classes are used.
2. Through Java class reflection.
3. Using Jini leases.
4. A JNI bridge to C++ is provided, but no general wire protocol is exposed for other platform support. JNI also incurs the overhead of a complete JVM per C++ application.
5. Tuples don't have traditional types. Standard matching requires tuples to have the same number of fields in the same order by type, but more flexible database style queries are supported.
6. Only C is implemented, but they have a language independent wire protocol.
7. Based on descriptions in their papers [111, 120, 121].
8. No routing fields within tuples are provided, but the system provides one tuplespace per device and a means of insuring a tuple is eventually made available in the tuplespace of some specific device.
9. A special ONCEPERTUPLE registration mode is available that insures each tuple will be retrieved only once.
10. This is inherent to their design goals.

Only Java is supported, but all communication is socket based, so assuming Java class structure is not relied upon heavily in their serialization it could be possible to support other platforms.

Table 18 - Tuplespace Extension Support in Modern Tuplespaces
(✓ = has feature, × = doesn't have feature, ~ = limited support of feature)

As can be seen, none of the systems has all of the features we found necessary, nor do they have the same features as one another. The table does not tell the whole story since most of the systems have additional features that are well suited for their domains but inappropriate for an interactive workspace.

8.4 Non-Tuplespace-based Coordination Systems

In this section, we present coordination systems that are not based on the tuplespace model, but nonetheless share some of the characteristics we found important for coordination in an interactive workspace. First, we present coordination systems specifically designed to support ubiquitous computing. Blackboard-based systems and systems that don't fit elsewhere in this discussion are presented next. Finally, a summary which shows the relation of the various non-tuplespace-based coordination systems to the Event Heap is given.

8.4.1 Ubiquitous Computing

This first set of coordination systems were all designed for ubiquitous computing in general. Coordination systems associated with a room-based ubiquitous computing project were discussed along with the specific project in Section 8.2.2. The four systems described here are Jini, UPnP (Universal Plug-and-Play), HAVi (Home Audio Video Interconnect) and One.World. The first three are commercial initiatives designed to address the problem of coordinating activity across the increasing number of devices that surround us in our everyday lives. The final system, One.World focuses on providing failure tolerance, easy check-pointing and application mobility between devices in a framework for ubiquitous computing applications.

The three commercial systems are perhaps better characterized as toolkits than coordination systems, as they provide for RMI, event notification on state changes, and, in the case of Jini, an extended tuplespace system. While including so many different coordination models within the toolkit insures that the systems can handle most types of coordination, it comes at the expense of larger client libraries, and making programmers choose which coordination model is best for their applications. Further, it makes it harder to adapt applications to work together since no two applications are guaranteed to be using the same combination of coordination mechanisms. We believe a simple yet powerful coordination system following a single model is more intuitive for programmers, provides more extensibility, and is more easily able to support different platforms than coordination systems that follow the "kitchen sink" approach.

Jini

Jini [141] is a coordination system from Sun Microsystems that is built on top of Java and most closely resembles the RMI-Rendezvous coordination system described in Chapter 4, Section 4.5.1. It is designed to provide a better way for Java programs to find and interact with one another when running on ad hoc collections of devices.

The Jini architecture is designed to minimize the administration needed for a large network of devices by providing a rendezvous mechanism for clients and services. Services register themselves with a Jini server through a multicast protocol, and then upload a set of attributes and the byte code needed by clients to contact them. Clients use a similar protocol to find the server and then request services meeting certain attribute requirements, receiving byte-code stubs for matching services. Unlike many models, clients then interact with the service by executing the byte-code from the rendezvous server. This byte-code conforms to the interface for the desired service type, and may contain arbitrary code for performing the actual communication with the service instance. The Jini architecture also has a distributed event system to notify applications

when changes occur in other objects on the network, and a lease system to allow applications to allow the system to expire devices that have been removed from the network.

Some of the key benefits that are described for Jini in [141] are:

- The Jini system allows upgrades and updates to be installed and used without requiring all devices on the network to be shut down.
- At the disadvantage of having to understand a common byte-code (Java), Jini gives the advantage of allowing the actual communication protocol between the proxy stub running on the client and the main service to be updated over time. This differs from CORBA, for example, where the protocol is locked in and must be known by the compiler at the time of compilation of both the client and the service.
- Jini is a small, simple set of conventions that allow the formation of a dynamic distributed system.
- Due to the typing, a sub-class with added features can be sent to a client requesting some interface. That client can use reflection and type-discovery in Java to discover and use those features.
- Jini strives for the minimal set of rules to allow clients and services to find each other and interact.

Although Jini could serve as a coordination model for an interactive workspace, it would have many of the drawbacks of RMI-Rendezvous systems mentioned in Chapter 4, Section 4.5.1. One of the most obvious restrictions of the Jini system is that you are tied to only using Java, which immediately causes problems when legacy applications need to be supported. Jini does come with an event system for notification of changes in object state, and comes with JavaSpaces [7], an extended-tuplespace system that will be discussed in Section 8.3. These two provide mechanisms for looser coupling than what is provided by the basic Jini model.

The Jini Architectural Overview [8] gives a more detailed, technical look at Jini.

Universal Plug and Play (UPnP)

The UPnP standard [63] from Microsoft allows devices to function in a networked environment just by being plugged into the local network. It provides for service discovery, invocation between devices/services, and a means for services to provide a URL to a web-browser viewable interface to themselves. Interfaces must be HTML-based to display in a browser, but pre-programmed interfaces for specific service types may be provided in controller devices.

There are a couple of requirements for all UPnP services. First, all devices must host an XML device description document. This includes a list of the services on the device. Each service consists of a state-table, which models the state of the service (for example, which song is currently playing on a CD player), a control server, which receives and executes remote invocations, and an event server, which publishes events to subscribers on state change.

The system was designed to use the following standard protocols:

- **TCP/IP, HTTP:** Both are used as transport protocols.
- **SSDP (Simple Service Discovery Protocol):** This allows discovery of services by client-polling and/or service beaconing. It also provides a mechanism for announcing departure, although all participants are also supposed to time-out data. It is not clear why they don't use SLPv2 [75], another Internet standard service discovery protocol.
- **GENA (Generic Event Notification Architecture):** A publish-subscribe event system. This is what services use to subscribe to and publish events about state-changes.
- **SOAP (Simple Object Access Protocol):** An RMI protocol that uses XML over HTTP for invocation. This allows it to deal better with firewalls and proxies than standard RMI protocols.

Like Jini, UPnP is most accurately described as an RMI-Rendezvous system, having the disadvantages described in Chapter 4, Section 4.5.1. Like Jini, it also provides a publish-subscribe event advertisement system to overcome some of the tight coupling problems in this type of system. One advantage it has over Jini is that the SOAP protocol works over HTTP and can be used with any language, so UPnP is not restricted to Java. Since there is no common byte-code format, however, the invocation protocol is fixed, making the system potentially less extensible.

HAVi

The home audio/video interoperability (HAVi) [98] architecture provides a set of standards for how home A/V equipment can be built to interoperate. It is being developed by a consortium of consumer electronics and home entertainment manufacturers. The goal of the architecture is to provide users a means of controlling many devices through a single device. For example, all components of a home theater system could be controlled through an interface on the TV screen. It is not designed to allow devices to coordinate with one another without direction by the user:

"APIs for application coordination and management are not standardized within HAVi, so detailed application control directly by the shell remains vendor specific."

The standard includes several levels of compliancy, with the highest level supporting the download of Java byte-code from remote devices to be used in controlling them. All devices in the system are represented by a device control manager (DCM) through which all services on the device are exposed. The DCM may run on the device, or run by proxy on a more capable device. There is an invocation system to trigger actions on other devices, and a pub/sub event mechanism to be notified when state changes happen in other devices in the system. The networking platform is IEEE 1394, and there are facilities for setting up A/V streams between components. For Java controllable devices there is a standard set of Java UI classes that can be used, or for use in less compliant devices, there is a standard device control interface language called DDI (data-driven interaction) that allows the specification of some standard widgets and bindings between the widgets and the appropriate actions to invoke on the service being controlled.

HAVi is best described as a hybrid of UPnP and Jini. It has Jini's reliance on Java, but allows standard interfaces to be used on non-Java devices. Aside from the drawbacks mentioned for UPnP and Jini, since HAVi is not designed to allow non-human coordination of devices and applications it would not suitable for general coordination in an interactive workspace.

One.World

The One.World system [72] is being developed as a general framework for ubiquitous computing as part of the Portolano project [60] at the University of Washington. They have a "programming for change" philosophy that has inspired the construction of their system—they assume that the environment will always be dynamic and that this dynamism must be accounted for in the design of the framework. Their system doesn't fit well into any of the coordination categories described in Chapter 4, Section 4.5.

In their system, applications have no threads of their own. Instead, an application consists of a specification of what events it wishes to receive and a set of event handlers to process those events. Events might be new input from a sensor, user input, or state changes made by other components. If an application needs to do recurring work, it can sign up for an idle event. All event handlers must return after a short amount of processing.

While applications can have internal state, any critical state is supposed to be stored in a tuplespace associated with the application. Multiple applications may also be grouped together to share a tuplespace, and the applications and tuplespaces may be hierarchically arranged. Unlike

with the Event Heap, the primary purpose of the tuplespaces is to store data as tuples of attributes and values rather than to provide event notification or data transfer.

Since applications have no running threads, event handlers only run for a short time, and critical state is stored in the tuplespace associated with the program, the One.World infrastructure can cause applications to become quiescent by buffering incoming events for the application and waiting until all running event handlers finish. At this point, the application can be check-pointed by storing the tuplespace. It can also be easily migrated by moving the tuplespace and switching delivery of buffered and future events to the new machine.

Among other applications, the One.World infrastructure has been used to implement one version of a technologically augmented biology lab called Labscape [25]. It consists of a computer at each workstation in the lab that prompt the user with the set of experimental steps for that station and provides a means to record the results (in some cases, the results are recorded automatically). In [24] they report that implementing Labscape using One.World was significantly easier than their first implementation, which was done using only sockets.

We believe that One.World is a promising system for insuring that future applications are highly portable and failure tolerant. Unfortunately, since applications must be written according to specific guidelines, these features both come at the price of supporting legacy applications, which we find unacceptable for interactive workspace environments. It might be possible, however, to provide a bridge between ensembles of One.World applications and other applications that are coordinating through the Event Heap to allow One.World applications to function in an iROS based interactive workspace.

8.4.2 Blackboard-based

The two coordination systems described in this sub-section fall in the category of "blackboard" systems, which are functionally almost identical to tuplespaces but were developed for work in artificial intelligence rather than parallel computing (more information on blackboard systems can be found in [58]).

The Hearsay-II Blackboard System

Hearsay II [59] was a non-real-time speech recognizer developed in the late 1970's that had about a 90% accurate recognition rate with a vocabulary of 1000 words. It was built on top of a blackboard metaphor that is similar to a tuplespace. Essentially a blackboard is a space where agents working on an artificial intelligence problem can "write" known information, or from which they can read or remove information in order to reason towards new results. The paper

also presents the blackboard system as a framework for attacking other AI problems beyond the speech recognition problem on which they were working.

The system was constructed out of knowledge sources (KS's), which were small programs. These programs were characterized by a condition and an action. The condition dictated under which circumstances the program was appropriate to be run. The action specified what would be done with the information (hypothesis) that triggered the condition to create a new hypothesis for the system. Conditions were triggered as KS's posted new hypotheses to the blackboard. Each new hypothesis was passed to the KS whose condition was triggered, which then processed it and posted the new derivative hypothesis or hypotheses to the blackboard.

In some ways these operations could be seen as programs doing a blocking *in* operation on a tuplespace, then processing the data and doing an *out* on the derivative information that was created. Unlike with traditional Linda-style tuplespaces where processes generally need to be authored together, KS's could be written by different programmers and combined in various ways to experiment with optimal combinations. All that needed to be agreed upon were the data formats for the hypotheses used in the blackboard.

Open Agent Architecture

The Open Agent Architecture (OAA) [107] is a system for building and coordinating communities of interacting agents developed by SRI. Agents are autonomous processes, and may be entirely computational, or an interface between a user and the rest of the agents. The system is based on a central facilitator to which all agents must register their capabilities. As time proceeds, goals may be expressed by agents (e.g. "Run voice recognition on this sound clip, then Fax it to Bob"), and the facilitator will handle dispatching sub-tasks to agents with the appropriate capabilities, building up partial results until the goal is met.

In addition to this basic mechanism, there is a state manager that allows registration of data either persistently or non-persistently to a globally accessible data store (much like a tuplespace or a blackboard). Triggers can be set on certain combinations of data in this "blackboard," and a new goal is created in the system when the trigger is activated. Triggers can also be set based on time, creation of goals, and several other things. This state management system is an evolved version of the blackboard used in the Hearsay-II system, which was just discussed.

One primary goal for the system is making it easy to integrate legacy applications. To that end, there is support for several different programming languages including Prolog, Lisp, Java, C, and C++.

The Open Agent Architecture comes from the Artificial Intelligence Center at SRI, so the design of the system is oriented toward solving AI problems. In large part the system is set up to allow problems to be intelligently solved by providing a modular way to put together agents, each with the ability to solve specific types of sub-problems. For example, the system is well suited to allowing a speech recognizer to convert speech to text, and then having the text be parsed by a grammar engine, which creates new goals to execute the various commands given in the original voice request.

Despite the different domain of application, many of the stated goals of the developers of the OAA system were quite similar to the ones we had in designing the Event Heap. They divided their goals into three different areas, interoperation and cooperation, human oriented user interfaces, and realistic software engineering requirements. Their goals were as follows:

- **Interoperation and cooperation:**
    - **Provide flexibility in assembling communities of autonomous service providers:** Autonomy is the hallmark of individual agents and this necessitates greater flexibility in interactions within communities of agents. This corresponds to our human centered interaction and flexible reconfiguration characteristic (**H2**).
    - **Provide flexibility in structuring cooperative interactions among agents:** The framework should provide for a variety of interaction patterns among agents. This corresponds to our support of all routing patterns feature (**F2**).
    - **Impose the right amount of structure on individual agents:** Allow message content to imply some meaning without being too rigid. Similar to our self-describing tuples (**F7**) and flexibly typed tuples (**F8**) features.
    - **Include legacy and "owned-elsewhere" applications:** Legacy applications are ones written before the agent system, and "owned-elsewhere" are those that are controlled by other institutions, and thus can't be modified. The goal is to allow integration of these "without requiring an overwhelming integration effort." This corresponds with one aspect of our heterogeneous software characteristic (**T2**).
- **Human-oriented user interfaces:**
    - **Conceptually natural:** Users should be able to express requests without needing detailed knowledge of the individual agents that will be involved.

- **Users as privileged members:** Human and computer agents should be able to cooperate.
- **Collaboration:** Allow users and agents to simultaneously work with data and processing resources.

- **Realistic software engineering requirements:**
  - **Minimize effort:** Should be easy to create new agents or to wrap existing applications. Similar to the Simple API feature (**F11**).
  - **Encourage reuse:** Agents constructed in the system should be reusable in different contexts. Similar to our application portability property (**P10**).
  - **Support lightweight, mobile platforms:** A massive local environment to support the system should not be required. Like our simple and portable API property (**P5**).
  - **Minimize platform and language barriers:** Use of the system should not require the adoption of a new language or environment. Similar to our heterogeneous devices and software characteristics (**T1** and **T2**).

They also make an argument for the use referential decoupling that is similar to ones that we make:

> "While it is possible to specify one or more agents using an address parameter (and there are situations in which this is desirable), in general it is advantageous to leave this delegation to the facilitator. Programming in this style greatly reduces the hard-coded dependencies among components that one often finds in other distributed frameworks."

Despite the many similarities in goals and in some of the arguments for features in the OAA system, the ideas for the Event Heap were developed independently. We believe that the similarities in arguments, and in some of the techniques used, helps to validate the Event Heap model.

The OAA system does differ, however, from the Event Heap in several ways. One difference is it's intended use for binding together mostly virtual agents, thus there is no real binding between agents and physical devices which they control. Another difference is that routing of information is controlled centrally by the facilitator, which needs to have some overall knowledge of capabilities of individual agents and end-goals. In contrast, the Event Heap server knows only how to match up posted data with the requests of receivers, allowing the receivers to determine

what they want to receive and when. OAA is also much more task oriented than the Event Heap. It is designed to allow users to specify some action to take place, and then allow the community of agents to satisfy that request. In contrast, the Event Heap is designed to allow collections of human accessible applications to coordinate with one another.

8.4.3 Other

Since the final two coordination system implementations were not specifically designed for ubiquitous computing and are not based on tuplespaces, they don't fit well into the previous two categories. The first system is the Information Bus, one of the most prominent publish-subscribe systems. The second is the Intentional Naming System, a content based routing system most similar to Message Oriented Middleware (see Chapter 4, Section 4.5.3).

Information Bus

The Information Bus [115] could reasonably be considered the canonical publish-subscribe system. As such, its function is similar to what was described in Chapter 4, Section 4.5.2, and it has the same advantages and disadvantages as a coordination infrastructure for an interactive workspace that were described there .

The Information Bus was designed for 24x7 uptime usage in real-world commercial settings. The system works over standard TCP/IP networks. Publishers send out data tagged by subject, and subscribers specify content of interest by specifying the subject they wish to receive, possibly using wildcards for partial specification. They also provide a RPC function where "subjects" are identified with callable processes. To invoke against that process a publish call is made to the channel with the appropriate "subject" and a special channel is used for the return values.

In designing the system, they had a set of three requirements and four design principles. The requirements were:

1. **Continuous Operation:** The system as a whole cannot be brought down of upgrades or maintenance.
2. **Dynamic System Evolution:** The system needs to support dynamic integration of new services and information. Changes in data format and application architecture must be tolerated. This is the same as our system evolution characteristic (**T4**).
3. **Legacy systems:** New software must be able to interact with old software. This is one aspect of our heterogeneous software characteristic (**T2**).

Their design principles were:

1. **Minimal Core Semantics:** Core communication system can make few assumptions about the semantics of the application programs to be coordinated. We have the expressiveness property (**P4**) for similar reasons.
2. **Self-describing Objects:** Both services and data objects are "self-describing" via a meta-object protocol which allows queries about type attributes, etc. This allows introspection. This is identical to our self-description feature (**F7**).
3. **Dynamic classing:** New classes can be dynamically defined and used to create instances, and principle two then allows these new classes to be used by older applications. This is similar to our flexible typing feature (**F8**).
4. **Anonymous communication:** Data objects sent and received are based on a subject not on specific IDs of senders and receivers [referential decoupling]. This allows subscribers and publishers to be introduced or removed at any time. This is what we call the referential decoupling property (**P2**).

One other similarity between the Information Bus and the Event Heap model is that they provide similar ordering guarantees. The Information Bus gives exactly once, FIFO ordering, with no ordering across senders, which is the same as what the Event Heap provides when registration is used (this is not surprising since registration is effectively a publish-subscribe mode in the Event Heap).

Intentional Naming System

The Intentional Naming System (INS) [19] provides a mechanism for routing network information based on the characteristics of the target, rather than an explicit name. For example, one could target a request to all printers on the third floor, instead of having to know the printer names. Attributes are chosen by the potential targets, and are hierarchical, so a given target might be in room 312, on the third floor, in Smith Hall, at City College, in Anytown, in the USA, etc.

There are two mechanisms for routing, early and late binding. Early binding functions like Domain Name Services (DNS) does for the Internet: a target or targets matching a set of attributes are requested, and explicit handles are returned and used in the future to route directly to those targets. In late binding, packets are sent out with the desired attributes and are passed among the Intentional Naming Routers until they reach a target or targets whose attributes match those which were specified.

Targets advertise their attributes and the routers use soft-state maintenance to insure correct state when targets go down. Overall, the system integrates name resolution and routing, allows resolvers to self-configure into an overlay network on top of an existing TCP/IP network, and uses soft-state mechanisms to provide weak consistency.

Their goal was to integrate name resolution and routing, while maintaining the following system properties:

- **Expressiveness:** Naming system must handle a wide range of devices and allow arbitrary services to be advertised, and arbitrary queries to be made. In our system this was property (**P4**).
- **Responsiveness:** Must adapt quickly to changing targets and mobile sources. Equivalent in their domain to perceptual instantaneity (**P7**) for coordination in interactive workspaces.
- **Robustness:** Resilient to router failures and inconsistent internal state. This is similar to our failure tolerance property (**P9**).
- **Easy configuration:** Minimal manual intervention to set up the system, and no manual registration of targets. Load should automatically be distributed across the routers/resolvers.

The designers of INS focused more on the routing mechanism than developing a full-fledged coordination system. Still, it is most similar to Message Oriented Middleware (MOM, discussed in Chapter 4, Section 4.5.3) since it allows routing of messages through several intermediate machines. The Event Heap by comparison is designed for coordination in a single room and typically works only on a single sub-net. Compared to the Event Heap, INS allows for similar content based routing, but provides a more sophisticated hierarchical schema for specifying targets. Unlike the Event Heap, INS does not allow receiver control over receiving content—all routing is specified by the sender. Just as the Event Heap tries to provide a better mapping between the scope for devices coordinating in an interactive workspace and the user's understanding of what should be interacting in the physical environment (the bounded environment interactive workspaces characteristic, **H1**), INS tries to provide a better mapping between the user's mental-model of their physical environment and the underlying routing mechanisms used by the systems infrastructure.

8.4.4 Summary

Table 19 summarizes the similarities and differences between the Event Heap and the various non-tuplespace-based coordination systems that have just been discussed.

Related Work	Similarities	Differences
Ubicomp Coordination		
Jini	• Goal of providing general coordination mechanisms for ubiquitous computing	• No provision for heterogeneous environments (Java only) • Tightly coupled RMI-based model
UPnP	• Goal of providing general coordination mechanisms for ubiquitous computing • Support for heterogeneous platforms	• Tightly coupled RMI-based model
HAVi	• Goal of providing general coordination mechanisms for ubiquitous computing	• Intended for facilitating human interaction with collections of consumer electronics • Tightly coupled RMI-based model Humans must drive application coordination
One.World	• Emphasis on providing for changing environments • "Design for failure" philosophy	• No heterogeneous support (Java only) • No legacy support (applications must use custom programming model)
Blackboard-based		
Hearsay-II	• Blackboard mechanism is similar to tuplespace model	• Designed for building up hypotheses from independent processes which encode AI principles • Not for general application coordination
Open Agent Architecture	Many goals similar to those for Event Heap: • Flexibility • Legacy support • Lightweight client infrastructure code • Support for heterogeneity	• Focused on coordinating collections of reasoning AI agents for intelligent processing • System relies on "smart" central controller that routes information as appropriate to insure progress towards goal
Other Coordination		
Information Bus	• Provide referential decoupling (publish-subscribe system) • Goal of providing legacy support • Emphasis on heterogeneous environments	• No temporal decoupling • Doesn't support anycast routing pattern
Intentional Naming System	• Provides system-level mechanism that mirrors human mental-model (route information based on "real-world" description of target)	• Focus only on routing

Table 19 – Comparison of Non-Tuplespace-Based Coordination Systems to the Event Heap

Chapter 9 – Open Questions and Future Work

Any significant research effort leaves certain aspects of the problem poorly addressed, and opens up new research problems that were not known before hand. This chapter describes some of the unresolved issues that remain after the characterization of interactive workspaces and the design and implementation of the Event Heap model. It also describes future research directions, some of which will be pursued within the Interactive Workspaces group after the conclusion of the research work presented in this dissertation.

9.1 Type Collision Issues

The issue of type collision arises in any system that allows interoperation of applications not designed to work together, so it follows that it is an issue for any coordination infrastructure designed to work in an interactive workspace. There are two main issues: name collision on event types and use of different event type names for similar information. Type collision occurs when two application designers choose the same event type name for events with different content. We expect that this will be relatively rare since we use strings for type names, making the namespace large. Further, since field names and types must also match for the events to match, unless the designers also choose the same set of mandatory fields, there shouldn't be a collision. Other discussion of typing and naming and how the two are inter-related for tuplespace-based systems such as the Event Heap can be found earlier in this dissertation in the flexible-typing part of Chapter 4, Section 4.2.3.

The second problem is more likely: two designers choose different type names for events conveying similar content. This could happen, for example, when two projector companies come up with different names for the event types that control their projectors. Fortunately, adapters

applications can be made using the interposability feature of the system. An adapter is a software intermediary that can pick up events of one type and convert them into events of another type that then get deposited in the Event Heap. Other researchers have done work on this in the past, for example with the Event Exchange system [110], which worked with T Spaces [145]. We have an initial version of an application called the "Patch Panel"[26], developed by Tico Ballagas (discussed in Chapter 6, Section 6.1.1), which allows these type conversions to be easily set up either programmatically through sending events to the Patch Panel application, or through a simple GUI.

The issue of collisions in type-names results from one of our underlying design philosophies. Specifically, we chose to make the system as open and extensible as possible, attempting to provide for the interaction of software components designed independently. One side-effect of this decision is that type-name collisions are a possibility. An alternative approach would have been to define a set of event types and semantics in advance and then to have enforced the use of these in the Event Heap. While this would have eliminated the type-name collision problem, it would have restricted the types of coordination supported by the Event Heap. We chose to trade what we believe is a small probability of name collision on types for a more open system.

9.2 Improving Performance and Integrating More Communication Types

As discussed in Chapter 7, Section 7.4, we did not envision an extended tuplespace-based architecture in general, or the Event Heap specifically, to be the only method of communication among applications and devices in an interactive workspace. Specifically, traffic that needs high throughput or low latency, or both, did not seem appropriate for this type of coordination system. In fact, our intention for the Event Heap was that it handle no more than a few events per second at a latency of around one hundred milliseconds per event. This led us to implement PointRight [87], our mouse redirection system for the iRoom, using the Event Heap to handle coarse scale configuration changes, and direct socket connections for transmission of actual mouse events.

Experience with our latest Event Heap implementation has led us to reconsider. As shown in Chapter 6, Section 6.3 even when handling several hundred events, and several hundred devices the latency remains below 50 ms. We now use the Event Heap for transportation of mouse events for the Macintosh port of PointRight, developed by Jeff Raymakers and Robert Brydon. This version allows several users to smoothly control the mouse over an 802.11b wireless network.

Based on the experience with PointRight we hope to define a broader set of coordination and communication types that can be handled through an extended tuplespace in the interactive

workspace domain. Our current implementation is not highly optimized, so we hope to improve performance without changing the current interface. We would also like to investigate adding new semantics for high throughput streams so that developers don't need to use multiple communication libraries to program applications for an interactive workspace.

9.3 Managing Application Coordination

As stated in Chapter 3, Section 3.4.3, one of the user needs in an interactive workspace is to be able to dynamically coordinate applications with one another. For example, if two different calendaring programs are brought up, it should be possible to link the applications such that selecting a new date in one application should cause the same date to be selected in the other. This should be possible even if the two applications were not designed to work together.

In the Event Heap, applications coordinate by emitting and consuming events of interest. Since applications are decoupled, the location where any given application runs is not important, and no application need be running that can respond to emitted events. While this technique works and has been successfully put to use in the CIFE Suite (see Chapter 6, Section 6.1.1), there are some problems that make it an incomplete solution. The first is that applications written by different developers may not use the same event types and fields. One company may call their date event 'ACMEDate' and the other just 'Date.' Further, one may encode the date as an integer of milliseconds since 1970, and the other in MM/DD/YYYY string. While an intermediary could be written to translate back and forth between the event formats, this is not a convenient mechanism for non-programmers trying to get applications running together in the workspace. We hope to investigate user interfaces that would allow users to associate applications in an intuitive fashion.

A second problem is that users may not want the two calendar programs to update their dates in synchrony. Perhaps they want to view their Tuesday schedule in one calendaring program and compare it to previous Tuesdays being displayed in the second program. Currently there is no mechanism either at the system or HCI level for users to specify which applications should coordinate their activities. One promising solution is to use the standard 'Group' field that has been designed into the Event Heap for routing purposes. Coordination could be constrained by setting the group field such that senders only post to the group with which they should coordinate and receivers only receive events from those in that same group. To do this would require some standard mechanism of changing the group field of any Event Heap client, and some UI to allow users to set up the groups.

These are two problems with dynamic application coordination that we hope to address in the future, but there are certainly challenges beyond these. In the end we would like to provide a system that allows users to easily select from diverse sets of applications and specify how they will interact with one another using techniques with low cognitive overhead that are suitable for collaborative problem solving environments.

9.4 Security and Privacy

In some ways, security and privacy in interactive workspaces are simplified by the use of the Event Heap model since authenticated users can be given access to the Event Heap and will be restricted to coordination in that space (this is another way tuplespaces fit well with the bounded environment characteristic of interactive workspaces, **H1**).

To date, however, we have not done a thorough investigation of the issue of security and privacy in interactive workspaces. The current situation is crude: iRooms are firewalled from the outside and reasonably physically secure, but once inside the room there is no other authentication or access control (except as required by specific applications, e.g. if an iRoom user attempts to open files from a remote fileserver). The situation is complicated by the fact that permanent devices use legacy-OS building blocks, such as Unix and Windows, and each assume (differing) single-user-at-a-time models for controlling the console display and allocating privileges. For now, a fictitious user with minimal privileges is permanently logged in at all the machines that control public infrastructure.

One major problem with security and privacy for Event Heap, or any tuplespace model, is that many of the useful features make the system inherently non-secure. Gelernter and Carriero acknowledged this in one of the earliest papers they published on tuplespaces [64]:

> "Properties such as distributed naming and sharing make Linda inherently less secure than other languages: Care must be taken to prevent unauthorized access to stored tuples."

The main problem is that security in general involves encrypting communication for privacy, thus making it opaque. This immediately breaks the extensibility of the system since it is no longer possible to observe events for reverse engineering. Interposition and snooping become impractical, and debugging is impaired. Further, the referential and temporal decoupling are potentially impaired if events are private to sources and receivers (since they need to know about each other for the encryption/decryption).

Another reason that security and privacy are challenging issues is that there is no clear social model for security in the interactive workspace domain. Unlike in a point-to-point interaction

where the evident security model is to perform some sort of encryption so that third-parties can't spy on the information, interactions in an interactive workspace are many-to-many. Further, users come into the space with the understanding that they will be working together, so there is already an implicit understanding that some if not all of the activities and information exchange that take place will be public to the participants.

We need to develop a set of scenarios to better understand what sorts of security are needed. Here are some potential example scenarios:

- A user moves information from a personal device to a large screen making it public, but before that it is kept private.
- Giving users the ability to control public infrastructure remotely from a personal device introduces a tradeoff between user convenience and authentication (as is typical in security related scenarios).
- Charts and photos can be shared at a meeting but copies may not be taken out of the room without permission from the owner.

In one of his early papers on ubiquitous computing [143], Weiser acknowledged that "it is important to realize there can never be a purely technological solution to privacy, and social issues must be considered in their own right."

Any security and privacy provisions added to the infrastructure must also take into account specific potential security needs within an interactive workspace (many of these needs were suggested for ubiquitous computing in general in [95]). Devices in a space may need protection from other devices brought in by visitors to the space. Users that bring in devices will want protection from other users' devices and the permanent devices in the workspace. Users will want some amount of privacy for material brought in on their laptop, PDA or other personal device. For impoverished devices, special security models may need to be employed. Finally, the needs of battery-powered devices may need to be accounted for: low power security routines and protection from battery-drain attacks may be needed.

Other good references for general security issues in ubiquitous computing applications are [97], which proposes a ubiquitous computing security framework, and [95], which looks at security as one of several issues in ubiquitous computing.

9.5 Beyond the Isolated Interactive Workspace

So far, the Interactive Workspaces project as a whole has dealt primarily with system infrastructure and HCI for a single interactive workspace. While we have always considered this to be "any" interactive workspace, in practice implementation has always been for the iRoom in Gates B23 at Stanford University. This is as true for the Event Heap implementation as for the rest of the software that has been developed. While no values have been hard-coded into the software that restrict it to use in the iRoom, little thought has been given so far to the issues that come up when you go beyond one interactive workspace in isolation.

One of these issues is how to provide for user sub-groups forming during a collaboration session within an interactive workspace. As discussed in Chapter 3, Section 3.2.2, in studies of team collaboration in project rooms, teams that were able to smoothly move back and forth between sub-groups and the large group were more efficient than those that only worked in a large group. When sub-groups form within one interactive workspace, each sub-group is effectively forming an interactive workspace within the interactive workspace. The question is how to accommodate the sub-groups by insuring applications and devices being used by a sub-group do not interact with other parts of the workspace. One possible solution is to use the 'group' field to tag events and restrict event reception as discussed in Section 9.3. Another is to create a sub-Event Heap for the sub-group as it forms, and then merge the sub-Event Heap back into the main Event Heap as the sub-group dissolves back into the main group.

Similar to the sub-group problem is the issue of dealing with poorly defined physical spaces. Characteristic **H1** of interactive workspaces was that they are physically bounded environments, and therefore to match user expectations the infrastructure should constrain devices and applications to only interact with other devices and applications in the same space (unless the user explicitly overrides this). While this principle holds for well-defined spaces like rooms, it is not clear if it applies for some other spaces. For example, in an office with multiple desks and occupants, should all devices interact, or should a device only interact with those of the same owner at the same desk? What about for large auditoriums with hundreds or thousands of people at a public event? Construction sites? Or, out in the middle of a forest? We don't have any good answer or solution for these situations, but as with sub-groups, some form of hierarchical Event Heaps may be the answer. Having users explicitly instantiate Event Heaps for ill-defined regions is another possibility.

Now that there are multiple locations on the Stanford campus that have deployed iROS and the Event Heap, roaming between interactive workspaces has become an issue for users with laptops.

To interact with other applications in an interactive workspace, a program connects to the Event Heap for that workspace. Currently this is done via a call to create a connection to some specific machine and server. Most applications are created to take the Event Heap server machine as a command line parameter, but it is inconvenient for users to start and stop each application with a new parameter every time they switch to a new interactive workspace. Further, if a user forgets to stop his applications and switch them to the new workspace, he may accidentally trigger activity in the interactive workspace he was in previously.

Currently the Macintosh OS X implementation of the Event Heap and iROS (done by Jeff Raymakers and Robert Brydon) uses the ZeroConf protocol to automatically discover Event Heap servers on the local sub-network. Users can then select the Event Heap server to which to connect in a preference pane. The software automatically stops and restarts all applications upon server changes, passing the name of the new Event Heap server in the appropriate command line argument. This is a good temporary solution, but some mechanism for switching Event Heap applications in a more standard fashion needs to be investigated.

We believe using ZeroConf, the Service Location Protocol [75], or some other system for discovering Event Heap servers on the local sub-net combined with a means to prompt users for the most appropriate server in ambiguous situations is probably a good solution for the short term. Defining spatial extents for workspaces and having devices use their location to select the appropriate Event Heap server would be a better solution, but will not be feasible until widespread, accurate location tracking is deployed.

Another problem that arises when you go beyond a single interactive workspace is the issue of remote participants. In particular, one of our collaborators, Oliver Brand at the Learning Lab of Lower Saxony (L3S) [9], would like remote learners to be virtually present in an interactive workspace for small seminar classes. While a simple solution is to allow remote participants to connect to the Event Heap for the interactive workspace where they wish to participate, this could allow them control over facilities in the room that should only be available to locals (printing to the room printer, for example). Some sort of access control or restriction of the type of events the remote learners are allowed to see is probably needed.

Finally, the issue arises of how to allow collaborators in one or more distinct tele-connected interactive workspaces to interact with one another. So far, one of our collaborators at KTH in Sweden, Erik Eliasson, has created an Event Heap bridge that can be configured to selectively filter events back and forth between two connected Event Heaps. We need more experience with the bridge to better understand what sorts of events are appropriate to pass between two

interactive workspaces (updates to shared data, perhaps), and which are inappropriate (light control events should probably stay local). Further, some sort of easy tool to allow users to configure the filtering of events between workspaces is also needed. Another issue that becomes critical in such multi-workspace interactions is security and privacy, which were discussed in Section 9.4.

9.6 Suggestions from Users

Both the developer and the administrator surveys that were presented in Chapter 6, Section 6.2 also contained requests for suggestions on how to improve the Event Heap. Those recommendations are presented here.

The developers had three main suggestions, two related to performance and one feature request. The first performance request was for special low latency paths within the Event Heap. These could be used for streaming control events such as mouse movements through the Event Heap while avoiding or reducing the overhead of performing matches on each event. The second performance request was that the size of the WireBundle packets that are sent over the TCP/IP connections be reduced. Since there are many standard fields all with string names, and WireBundles are uncompressed, the smallest possible packet size is around 700 bytes. When hundreds of events per second are being streamed through the server, with two hops on the network each, the traffic can begin to saturate 11 Mbps wireless links. As mentioned in Chapter 7, Section 7.4, the Event Heap was originally intended for infrequent coordination between applications, so low latency paths and small packets were not originally a priority. Since the current version of the Event Heap is much faster than the earlier versions, users have begun streaming mouse events and other data for UI interaction that is latency sensitive. A desire for even better support for this type of event traffic led to the developer requests for fast paths and smaller packet sizes.

The final suggestion from developers was a feature request to support RPC over the Event Heap. This indicates a need to use RPC style coordination within interactive workspaces. While an RPC can be implemented using a put and blocking retrieve on the caller and a blocking retrieve followed by a put on the callee, it can be annoying to developers to constantly have to recode this. Based on the survey, multicast-style coordination is the most common type in interactive workspaces, but evidently RPC is done frequently enough that it is a requested feature. Our opinion is that this should not be provided at the level of the Event Heap, but rather in a supplemental toolkit built on top of the Event Heap. This would allow the Event Heap client to

remain simple, while permitting applications that needed the RPC mechanism to have access to the appropriate routines.

Administrators had two main requests. First, they wanted the capability to switch interactive workspaces more quickly. Specifically this means the ability to switch the connected Event Heap server quickly. This is not surprising since most of the survey participants help to administer multiple workspaces, and they currently have to edit configuration files every time they move. This feature has already been discussed as future work in Section 9.5.

The second administrator request was for better security, and it makes sense that this would be a bigger concern to an administrator than to a developer. We acknowledge that security is one area to which we have not paid sufficient attention, and it was discussed earlier in the chapter in Section 9.4.

One request that was made by both developers and administrators was that the Event Heap support prioritization of events. In both cases the request was framed as a way of insuring that bursty, long-lived, high bandwidth traffic, like transfers of video files, would not interfere with the latency of UI events which need to have low latency in order to insure perceptual instantaneity (**P7**). Unfortunately, this is impossible to do at the level of the Event Heap itself, since there is no API at the socket level for requesting that other network traffic be throttled. This problem could be solved by deploying traffic prioritization mechanisms at the data link or network layers that give priority to Event Heap traffic within each interactive workspace. The Event Heap code could then be modified to provide prioritization of its own for applications coordinating through it.

9.7 A General Purpose System Model for Ubiquitous Computing

One question that can be asked of any systems framework is whether it could be made more general to support computing beyond its original scope. While the Event Heap was only designed to support coarse-grained inter-application coordination in an interactive workspace, it is already being used for sending low-latency streams of mouse events. Within the iROS system, we also have the Data Heap (discussed in Chapter 6, Section 6.1.1) which stores persistent data tagged with a set of attributes. Currently the Data Heap has a separate API that works on top of the Event Heap, but since events are also a collection of attributes (fields), it may be possible to extend the capabilities of the Event Heap so that it can handle persistence directly. We also have a soft state management system written on top of the Event Heap that beacons sets of attributes to allow for state in an interactive workspace (this complements events which typically signal a

change in state). Since this is also attribute based, it could potentially be supported in the base Event Heap as well. While giving the Event Heap more functionality within a similar basic framework is appealing since developers only need to learn one system, this comes with the drawback of a more complicated implementation, which would restrict the portability of the Event Heap client API to new platforms. Any changes would have to be carefully examined to determine whether the cost outweighed the benefit.

The iROS system and the Event Heap, in particular, have been designed as infrastructure to support applications and devices in an interactive workspace. The domain of ubiquitous computing is much broader, however, and it remains a question how to support applications for this domain in a more general sense.

One obvious solution is to somehow extend the Event Heap model to apply in other ubiquitous computing situations: ad hoc meetings of people with their personal devices, in cars, in museums, roaming around cities, etc. Due to scalability issues with centralized exchange mechanisms, it is not obvious how to do this. One possible mechanism would be to allow hierarchical Event Heaps. Devices could have local Event Heaps which get joined to form an Event Heap for a person's Personal Area Network (PAN), and those could get joined together with other people's PAN's in ad hoc situations, or to permanent Event Heaps associated with fixed interactive workspace installations.

As was shown in 0, there have been many middleware systems designed and implemented which provide coordination infrastructure for ubiquitous computing. It would be intellectually pleasing if there were some ideal infrastructure, perhaps a hybrid of several of those proposed, that was simple yet capable of handling the full variety of activities that can transpire in ubiquitous computing scenarios. A study that evaluates the appropriateness of the various proposed infrastructures could provide a better perspective on the feasibility of such a universal coordination system for the domain. Since the field of ubiquitous computing is relatively new, however, there is no canonical set of relevant scenarios, established collection of ubiquitous computing applications, or well-agreed upon set of metrics for evaluating how well an infrastructure supports any given application or scenario. Coming up with scenarios, applications and metrics, and using them to do a more formal comparison of proposed infrastructures will be an important task for the ubiquitous computing community over the coming years.

Much as the World Wide Web accelerated the development and use of the Internet, perhaps a unified, simple, and extensible middleware system to support ubiquitous computing applications and scenarios is what is needed to accelerate the development and use of ubiquitous computing.

Chapter 10 – Conclusion

The Interactive Workspaces group at Stanford University has been investigating how to provide the software tools and human computer interaction techniques necessary for ubiquitous computing in the specific sub-domain of ubiquitous computing rooms (which we call interactive workspaces). These environments (described in detail in Chapter 1) are technologically rich team project rooms where groups of users can come together to collaboratively solve problems with the assistance of computational power in the environment.

This dissertation has addressed the need for software tools and infrastructures to permit the designing and running sets of applications that coordinate with one another within an interactive workspace. Specifically, it presents the Event Heap coordination model for interactive workspaces along with an implementation of that model. There are three major contributions to the body of knowledge contained in this dissertation:

- **C1:** A characterization of interactive workspaces from the standpoint of factors that must be taken into account by any coordination infrastructure in such a space. This is in the form of a set of characteristics of interactive workspaces and was presented in Chapter 3.
- **C2:** The Event Heap, an infrastructure model for application coordination in an interactive workspace. The model has two parts: First, a set of properties that a coordination infrastructure needs to have to provide for the characteristics laid out in **C1**, and second, an extended tuplespace model whose features supply the needed properties. These properties and features and their motivation were presented in Chapter 4. While most of the properties and features have been implemented elsewhere, the appropriate combination for an interactive workspace had not been previously derived or implemented.

- **C3:** Instantiation of the Event Heap model as a working piece of infrastructure software, and demonstration that it serves its purpose through its use in dozens of applications running in approximately ten different prototype workspaces. The implementation was presented in Chapter 5.

In Chapter 6 we evaluated the validity of the characteristics of interactive workspaces, the Event Heap model and the model implementation. This was done by listing written applications and deployments, presenting the results of a user survey whose participants verified the utility of the system, and showing that the prototype implementation has sufficient performance for our domain. While we are confident that we have compellingly demonstrated the validity of the Event Heap model, as with many systems whose final goal is to provide a better user experience, until the system has proven useful in wide deployment, no conclusive proof will be available.

In [35], Carriero and Gelernter talk about how it was hard to quantify the usefulness of the original tuplespace model as implemented in Linda: "The work of systems researchers is significant only insofar as it's useful to non-systems-researchers." They argued that they would be satisfied if Linda met the following three criteria:

1. Linda is being used to solve "real problems".
2. The Linda solutions to these problems are easy to understand.
3. The Linda solutions demonstrate real speedup.

We believe that these are reasonable measures, and that comparing the Event Heap and its implementation to similar criteria yield positive answers. The Event Heap is being used by researchers both within our group and without, in computer science and in other fields, to implement and solve real problems. As the results of the user survey suggest, these solutions are easier to understand and simpler to implement than solutions in other systems people have used. Finally, while sheer speed is not as big an issue for us as it was for Carriero and Gelernter in the parallel programming domain, the current Event Heap implementation has proven to be several times faster than older Event Heap implementations based on others' technology.

We also believe that having a working implementation suitable for wide deployment in the research community has value beyond just validating the Event Heap model itself. While provision of development tools in and of themselves is not a research contribution, it is important to the continued growth of the ubiquitous computing field and facilitates future research. In his original article on the topic [142], Weiser posited that by the end of the 1990s there would be cheap low power computers with convenient displays, software to run those devices, but no clear

programming model to provide for cross device coordination. These predictions, for the most part, have come to pass. Laptops, PDAs and tablet computers are all now available, and they have their own operating systems and applications, but there is still no commonly understood method for cross device coordination. Other projects, including the Intelligent Room project at MIT [46], and the i-Land project at IPSI [136], have also found development of ubiquitous computing infrastructure to be a critical need for ubiquitous computing rooms.

Everyone carries around mental-models of the world around them. Any user's ability to interact with a computing environments is made easier to the extent that the underlying computation model aligns with his or her mental-model of that environment. For example, Carriero and Gelernter wrote the following about mental-models and programming languages [36]:

> "A useful programming language makes it easy for the programmer to embody a mental model directly in working code. A program that embodies a mental model simply and directly is easier to write initially, and easier to understand once it exists as a working application."

In their case, the programming language needed to align with the mental-models of parallel programs that developers needed to instantiate. Similarly, the computing environment of an interactive workspace needs to align with people's mental-models of such spaces. In these environments, however, there are two types of users—end-users and developers. To minimize unnecessary computer interaction overhead for end-users, the applications running in the space need to function and interact according to the user's mental-model of how programs should interact in the space. For developers there is an extra level of indirection; the development system must fit the mental-model of developers who are writing applications that fit the mental-model of end-users.

We attempt to align the both the end-user's and the developer's mental-models with the function of the Event Heap by providing for marketplace-style coordination associated with a specific, bounded environment. The former mimics how humans might interact in a room together, and thus how they might expect applications to behave. The latter mirrors the user's expectation that, just as human's normally can only interact with others in their local environment, devices and applications should also interact by default only with other devices and applications in their local environment. We hope that by mimicking user's expectations of how applications should function in an interactive workspace, the Event Heap will enable the development of new applications and the wider use and deployment of these spaces in the future.

While many principles influenced the design of the Event Heap, perhaps the most important one was decoupling. The Event Heap model inherits referential decoupling and temporal decoupling

from the tuplespace model upon which it is based (although we limit temporal decoupling in the Event Heap to avoid having to provide long-term persistence). Both types of decoupling contribute to the failure tolerance property of the Event Heap. The referential decoupling helps with by minimizing the interdependencies of applications, while temporal decoupling helps to mask transient failures. Referential decoupling also makes it possible to use intermediation for application adaptation. We believe the combination of failure tolerance and interoperability we have been able to realize would not have been practical using any other mechanisms.

As has been noted, few features are present in the Event Heap that have not been included before in at least one coordination infrastructure. The contribution of this dissertation lies in the synthesis of research on team-project rooms (from the field of Sociology), Groupware (from the field of HCI and the sub-field of CSCW), and construction of coordination systems (from the Systems research area of Computer Science). Information from these various fields was combined, along with experiential knowledge, to come up with the characteristics of interactive workspaces: bounded environment, human-centered interaction, human level performance needs, heterogeneous software and hardware, and short and long term change in the environment. Based on these characteristics, and an investigation of coordination systems intended for different domains, we were able to arrive at the set of properties and features that best define a coordination infrastructure specifically for interactive workspaces. Before this dissertation, there existed no good model for developing applications that could coordinate with one another in interactive workspaces subject to the constraints imposed by the characteristics of such spaces; now through our efforts we were able to develop the Event Heap model and implementation, which provides such a system for coordination in such spaces.

Finally, we close with the philosophy that underlies this dissertation and much of the other infrastructure work that is being done in the Interactive Workspaces project. Specifically, we choose to design systems for flexibility and future extensibility. The alternative is to design systems with pre-agreement from all parties of how interaction will take place. The latter approach has some advantages:

- Naming collisions are eliminated since all types and names are decided at design time.
- Compatibility of systems from different designers is facilitated since types and their meanings are pre-defined.
- The system is easier to optimize since the usage mechanisms are known.

- It is easier to formally analyze the system and the interaction of components that use the system.

On the other hand, by designing for flexibility and extensibility instead, the system gains the following:

- It can adapt over time.
- The system's designers don't need to have anticipated all usage scenarios.
- It is easier to adapt legacy applications to be compatible with the system.
- Future interoperability of applications from independent developers can be provided without the system being overly constraining.

While some of the features of pre-defined systems are useful, we believe that emphasizing flexibility and extensibility for systems to be deployed in the real world is critical to the long-term success of those systems. We hope that this belief will be borne out in the wide deployment and use of the Event Heap system.

THIS PAGE LEFT INTENTIONALLY BLANK.

List of References

1. *The American Heritage dictionary*. 2nd college ed. 1982, Boston: Houghton Mifflin. 1568.
2. *GigaSpaces Platform*. White Paper, . 2002, New York, NY, USA: GigaSpaces Technologies Ltd. 14. http://www.j-spaces.com/download/GigaSpacesWhitePaper.pdf (Verified 10/2002).
3. *Gryphon Project Home Page*, , IBM Research http://www.research.ibm.com/gryphon/home.html (Verified: 10/2002).
4. *Ideo Corporation*, http://www.ideo.com/.
5. *iROS Meta-Operating System*, . 2001-2002, Interactive Workspaces Group, Stanford University: Stanford, CA http://iros.sourceforge.net.
6. *iSpaces Project*, http://2g1319.ssvl.kth.se/2001/projects/iSpace.html.
7. *JavaSpaces Service Specification*, . 2000, Palo Alto, CA: SUN Microsystems. http://java.sun.com/products/javaspaces (Verified: 10/2002).
8. *Jini Architectural Overview*. Technical White Paper, . 1999, Palo Alto, CA, USA: Sun Microsystems. http://www.sun.com/jini/whitepapers/architecture.html.
9. *Learning Lab of Lower Saxony*, http://www.learninglab.de (Verified: 11/12).
10. *Macromedia Flash*, . 1995-2002, Macromedia, Inc.: San Francisco, CA http://www.macromedia.com/software/flash/.
11. *MIT Project Oxygen : Software Environment*, . 2001, MIT Laboratory for Computer Science, MIT AI Laboratory http://oxygen.lcs.mit.edu/Software.html.
12. *MQSeries: Message Oriented Middleware*, : IBM. http://www-3.ibm.com/software/ts/mqseries/library/whitepapers/mqover/ (Verified: 10/2002).
13. *SMART Board*, , SMART Technologies http://www.smarttech.com.
14. *Stanford Learning Lab*, http://sll.stanford.edu.
15. Abiteboul, S., R. Hull, and V. Vianu, *Foundations of databases*. 1995, Reading, Mass.: Addison-Wesley. xviii, 685.

16. Abowd, G., *et al. Teaching and learning as multimedia authoring: the classroom 2000 project*. in *Fourth ACM International Conference on Multimedia*. 1996. Boston, United States: Association for Computing Machinery.

17. Abowd, G., J. Brotherton, and J. Bhalodia. *Classroom 2000: a system for capturing and accessing multimedia classroom experiences*. in *CHI 98: Human Factors in Computing Systems*. 1998. Los Angeles, CA USA: Association for Computing Machinery.

18. Adams, E.N., *Optimizing preventive service of software products.* IBM Journal of Research and Development, 1984. **28**(1): p. 2-14.

19. Adjie-Winoto, W., *et al.*, *The design and implementation of an intentional naming system.* Oper. Syst. Rev. (USA), Operating Systems Review, 1999. **33**: p. 186-201.

20. Ahuja, S., N. Carriero, and D. Gelernter, *Linda and friends.* Computer, 1986. **19**(8): p. 26-34.

21. Anderson, J.R., *Learning and memory : an integrated approach*. 2nd ed. 2000, New York: Wiley. xviii, 487.

22. Applegate, L.M., B.R. Konsynski, and J.F. Nunamaker. *A group decision support system for idea generation and issue analysis in organization planning*. in *First Conference on Computer-Supported Cooperative Work*. 1986. Austin, Texas.

23. Arias, E., *et al.*, *Transcending the individual human mind-creating shared understanding through collaborative design.* ACM Transactions on Computer-Human Interaction, 2000. **7**(1): p. 84-113.

24. Arnstein, L., *et al. Systems Support for Ubiquitous Computing: A Case Study of Two Implementations of Labscape*. in *Pervasive 2002*. 2002. Zurich, Switzerland: Springer Verlag.

25. Arnstein, L.F., S. Sigurdsson, and R. Franza, *Ubiquitous Computing in the Biology Laboratory.* Journal of the Association of Laboratory Automation (JALA), 2001. **6**(1).

26. Ballagas, R., *et al. iStuff: A Physical User Interface Toolkit for Ubiquitous Computing Environments*. in *CHI 2003: Human Factors in Computing Systems*. 2003. Fort Lauderdale, FL, USA: Association for Computing Machinery.

27. Banavar, G., *et al. A case for message oriented middleware*. in *DISC'99: 13th International Symposium on Distributed Computing*. 1999. Bratislava, Slovakia: Berlin, Germany : Springer-Verlag, 1999.

28. Bernstein, P.A. and E. Newcomer, *Principles of transaction processing*. Morgan Kaufmann series in data management systems. 1997, San Francisco: Morgan Kaufmann Publishers. xxiv, 358.

29. Bisiani, R. and A. Forin, *Multilanguage parallel programming of heterogeneous machines.* IEEE Transactions on Computers, 1988. **37**(8): p. 930-45.

30. Bjornson, R., N. Carriero, and D. Gelernter. *From weaving threads to untangling the web: a view of coordination from Linda's perspective*. in *COORDINATION 97. Second International Conference on Coordination Models and Languages*. 1997. Berlin, Germany: Berlin, Germany : Springer-Verlag, 1997.

31. Blair, G., *et al. Quality of Service Support in a Mobile Environment: An Approach Based on Tuple Spaces*. in *5th IFIP International Workshop on Quality of Service (IWQoS'97)*. 1997. New York, New York, USA: Kluwer Academic Publishers.

32. Bollella, G., S. Graham, and T.J. Lehman. *Real-time TSpaces*. in *IECON'99. Conference Proceedings. 25th Annual Conference of the IEEE Industrial Electronics Society*. 1999. San Jose CA USA: Piscataway, NJ, USA : IEEE, 1999.

33. Borchers, J., *et al.*, *Stanford Interactive Workspaces: A Framework for Physical and Graphical User Interface Prototyping*, in *IEEE Wireless Communications, Special Issue on Smart Homes (in press)*. 2002

34. Brumitt, B., *et al. EasyLiving: technologies for intelligent environments*. in *Handheld and Ubiquitous Computing Second International Symposium HUC 2000*. 2000. Bristol, UK: Berlin, Germany : Springer-Verlag, 2000.

35. Carriero, N. and D. Gelernter. *Applications experience with Linda (programming language)*. in *ACM/SIGPLAN PPEALS 1988 - Parallel Programming: Experience with Applications, Languages and Systems*. 1988. New Haven, CT, USA: Usa : 1988.

36. Carriero, N. and D. Gelernter, *Linda in context (parallel programming).* Communications of the ACM, 1989. **32**(4): p. 444-58.

37. Carriero, N.J., *et al.*, *The Linda alternative to message-passing systems.* Parallel Computing, 1994. **20**(4): p. 633-55.

38. Carzaniga, A., D.S. Rosenblum, and A.L. Wolf. *Achieving scalability and expressiveness in an Internet-scale event notification service*. in *Nineteenth Annual ACM SIGACT-*

SIGOPS Symposium on Principles of Distributed Computing (PODC). 2000. Portland, OR, USA: New York, NY, USA : ACM, 2000.

39. Cerqueira, R., *et al. Gaia: A Development Infrastructure for Active Spaces*. in *Ubitools Workshop at Ubicomp 2001*. 2001. Atlanta, GA.

40. Chen, G. and D. Kotz. *SOLAR: Towards a Flexible and Scalable Data-Fusion Infrastructure for Ubiquitous Computing*. in *UbiTools '01 Workshop at Ubicomp 2001*. 2001. Atlanta, GA.

41. Chen, M. *Design of a Virtual Auditorium*. in *Ninth ACM International Conference on Multimedia*. 2001. Ottawa, Canada: ACM Press: New York, NY, USA.

42. Cherry, E.C., *Some Experiments Upon the Recognition of Speech, with One and Two Ears.* Journal of the Acoustical Society of America, 1953. **25**: p. 975-979.

43. Chou, T.C.K., *Beyond fault tolerance.* Computer, 1997. **30**(4): p. 47-9.

44. Ciancarini, P., *et al. PageSpace: an architecture to coordinate distributed applications on the Web*. in *Fifth International World Wide Web Conference*. 1996. Paris, France: Netherlands : Elsevier, 1996.

45. Clark, D.D. *The design philosophy of the DARPA Internet protocols*. in *SIGCOMM '88 Symposium: Communications Architectures and Protocols*. 1988. Stanford CA USA: Usa : 1988.

46. Coen, M.H. *A prototype intelligent environment*. in *Cooperative Buildings Integrating Information, Organization, and Architecture First International Workshop CoBuild'98 Proceedings*. 1998. Darmstadt, Germany: Berlin, Germany : Springer-Verlag, 1998.

47. Coen, M.H., *et al. Meeting the Computational Needs of Intelligent Environments: The Metaglue System*. in *MANSE99 : 1st International Workshop Managing Interactions in Smart Environments*. 1999. Dublin, Ireland.

48. Cooperstock, J.R., *et al.*, *Reactive environments.* Communications of the ACM, 1997. **40**(9): p. 65-73.

49. Cooperstock, J.R., *et al. Evolution of a reactive environment*. in *ACM Conference on Human Factors in Computing Systems*. 1995. Denver, CO, USA: New York, NY, USA : ACM, 1995.

50. Covi, L.M., *et al. A room of your own: what do we learn about support of teamwork from assessing teams in dedicated project rooms?* in *Cooperative Buildings Integrating Information, Organization, and Architecture First International Workshop CoBuild'98 Proceedings*. 1998. Darmstadt, Germany: Berlin, Germany : Springer-Verlag, 1998.

51. Curtis, B., *Modeling Coordination from Field Experiments*, in *Organizational Computing, Coordination and Collaboration: Theories and Technologies for Computer-Supported Work*. 1989: Austin, TX.

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

53. De Lara, E., D.S. Wallach, and W. Zwaenepoel. *Puppeteer: component-based adaptation for mobile computing*. in *3rd USENIX Symposium on Internet Technologies and Systems*. 2001. San Francisco, CA, USA: Berkeley, CA, USA : USENIX Assoc, 2001.

54. Dey, A.K., D. Salber, and G.D. Abowd, *A Conceptual Framework and a Toolkit for Supporting the Rapid Prototyping of Context-Aware Applications.* Human-Computer Interaction (HCI) Journal, 2001. **16**.

55. Dourish, P., *et al.*, *Presto: an experimental architecture for fluid interactive document spaces.* ACM Transactions on Computer-Human Interaction, 1999. **6**(2): p. 133-61.

56. Edwards, W.K. and R. Grinter. *At Home with Ubiquitous Computing: Seven Challenges*. in *Ubicomp 2001*. 2001. Atlanta, GA, USA.

57. Ellis, C.A., S.J. Gibbs, and G.L. Rein, *Groupware: some issues and experiences.* Communications of the ACM, 1991. **34**(1): p. 39-58.

58. Engelmore, R.S. and T. Morgan, *Blackboard systems edited by Robert Engelmore, Tony Morgan*. Insight series in artificial intelligence. 1988, Wokingham, England ; Reading, Mass.: Addison-Wesley. xviii, 602 , [4] of plates.

59. Erman, L.D., *et al.*, *The Hearsay-II speech-understanding system: integrating knowledge to resolve uncertainty.* Computing Surveys, 1980. **12**(2): p. 213-55.

60. Esler, M., *et al. Next century challenges: data-centric networking for invisible computing. The Portolano Project at the University of Washington*. in *5th Annual Joint ACM/IEEE International Conference on Mobile Computing and Networking (MOBICOM'99)*. 1999. Seattle WA USA: New York, NY, USA : ACM, 1999.

61. Eugster, P., *et al.*, *The Many Faces of Publish/Subscribe*. Technical Report, MSR-TR-2001-104. 2001, Cambridge, UK: Microsoft Research Laboratories. 24. http://research.microsoft.com/users/annemk/papers/tr01_004.ps (Verified: 10/2002).

62. Falkowski, C. *A room management system*. in *Cooperative Buildings Integrating Information, Organization, and Architecture First International Workshop CoBuild'98 Proceedings*. 1998. Darmstadt, Germany: Berlin, Germany : Springer-Verlag, 1998.

63. Fout, T., *Universal Plug and Play in Windows XP*. White Paper, . 2001, Redmond, WA: Microsoft Corporation. 32. http://www.microsoft.com/WINDOWSXP/pro/techinfo/howitworks/upnp/default.asp.

64. Gelernter, D., *Generative communication in Linda.* ACM Transactions on Programming Languages and Systems, 1985. **7**(1): p. 80-112.

65. Gelernter, D. and N. Carriero, *Coordination languages and their significance.* Communications of the ACM, 1992. **35**(2): p. 97-107.

66. Gelernter, D., *et al. Parallel programming in Linda*. in *1985 International Conference on Parallel Processing*. 1985. St. Charles IL USA: Washington, DC, USA : IEEE Comput. Soc. Press, 1985.

67. Gelernter, D. and L. Zuck. *On what Linda is: formal description of Linda as a reactive system*. in *COORDINATION 97. Second International Conference on Coordination Models and Languages*. 1997. Berlin, Germany: Berlin, Germany : Springer-Verlag, 1997.

68. Goland, Y., *et al.*, *HTTP Extensions for Distributed Authoring - WEBDAV*. Request For Comments (RFC), RFC-2518. 1999: Internet Engineering Task Force. http://www.webdav.org.

69. Gray, J. *Why do computers stop and what can be done about it?* in *Fifth Symposium on Reliability in Distributed Software and Database Systems*. 1986. Los Angeles CA USA: Washington, DC, USA : IEEE Comput. Soc. Press, 1986.

70. Greenberg, S., M. Boyle, and J. Laberge, *PDAs and shared public displays: making personal information public, and public information personal.* Personal Technologies, 1999. **3**(1-2): p. 54-64.

71. Greif, I., *Computer-supported cooperative work : a book of readings*. 1988, San Mateo, Calif.: Morgan Kaufmann. viii, 783.

72. Grimm, R., *et al. A System Architecture for Pervasive Computing*. in *9th ACM SIGOPS European Workshop*. 2000. Kolding, Denmark.

73. Guimbretière, F., *Fluid Interaction for High Resolution Wall-size Displays*. Ph.D. Dissertation, Computer Science. 2002, Stanford, CA, USA: Stanford University. 140.

74. Guimbretière, F., M. Stone, and T. Winograd, *Fluid Interaction with High-resolution Wall-Size Displays.* UIST (User Interface Software and Technology): Proceedings of the ACM Symposium, 2001: p. 21-30.

75. Guttman, E., *Service location protocol: automatic discovery of IP network services.* IEEE Internet Computing, 1999. **3**(4): p. 71-80.

76. Haartsen, J., *et al.*, *Bluetooth: vision, goals, and architecture.* Mobile Computing and Communications Review, 1998. **2**(4): p. 38-45.

77. Harper, R.R., J.A. Hughes, and D.Z. Shapiro. *Harmonious Working and CSCW: Computer Technology and Air Traffic Control*. in *First European Conference on Computer-supported Cooperative Work*. 1989. Gatwick, London, UK.

78. Hodes, T.D. and R.H. Katz. *A document-based framework for Internet application control*. in *USENIX'99: 2nd Symposium on Internet Technologies and Systems*. 1999. Boulder CO USA: Berkeley, CA, USA : USENIX Assoc, 1999.

79. Hughes, J., J. O'Brien, and T. Rodden. *Understanding technology in domestic environments: lessons for cooperative buildings*. in *Cooperative Buildings Integrating Information, Organization, and Architecture First International Workshop CoBuild'98 Proceedings*. 1998. Darmstadt, Germany: Berlin, Germany : Springer-Verlag, 1998.

80. Hull, R., P. Neaves, and J. Bedford-Roberts. *Towards situated computing*. in *First International Symposium on Wearable Computers*. 1997. Cambridge MA USA: Los Alamitos, CA, USA : IEEE Comput. Soc, 1997.

81. Humphreys, G., *et al.*, *WireGL: A scalable graphics system for clusters.* Proceedings of the ACM SIGGRAPH Conference on Computer Graphics, 2001: p. 129-140.

82. Ionescu, A., M. Stone, and T. Winograd, *WorkspaceNavigator: Capture, Recall and Reuse using Spatial Cues in an Interactive Workspace*. Technical Report, TR2002-04. 2002, Stanford, CA, USA: Stanford University. 8. http://www.stanford.edu/~arna/persist/wkspcNav-286.pdf Verified: 11/2002).

83. Johansen, R., *Leading business teams : how teams can use technology and group process tools to enhance performance*. 1991, Reading, Mass.: Addison-Wesley. xxiv, 216.

84. Johanson, B. and A. Fox. *The Event Heap: a coordination infrastructure for interactive workspaces*. in *Fourth IEEE Workshop on Mobile Computing Systems and Applications*. 2002. Callicoon, NY, USA: Los Alamitos, CA, USA : IEEE Comput. Soc, 2002.

85. Johanson, B. and A. Fox, *Extending Tuplespaces for Coordination in Interactive Workspaces.* To appear in Journal of Systems and Software, 2003(Special Issue on Application Models and Programming Tools for Ubiquitous Computing).

86. Johanson, B., A. Fox, and T. Winograd, *The Interactive Workspaces project: experiences with ubiquitous computing rooms.* IEEE Pervasive Computing, 2002. **1**(2): p. 67-74.

87. Johanson, B., *et al. PointRight: Experience with Flexible Input Redirection in Interactive Workspaces*. in *ACM Symposium on User Interface Software and Technology (UIST-2002)*. 2002. Paris, France.

88. Johanson, B., *et al. Multibrowsing: Moving Web Content across Multiple Displays*. in *Ubicomp 2001*. 2001. Atlanta, GA, USA.

89. Johnson-lenz, P. and T. Johnson-lenz, *Groupware: The process and impacts of design choices*, in *Computer-mediated communication systems : status and evaluation*, E.B. Kerr and S.R. Hiltz, Editors. 1982, Academic Press: New York, N.Y.

90. Jones, M.B., *Interposition agents: transparently interposing user code at the system interface.* Oper. Syst. Rev. (USA), Operating Systems Review, 1993. **27**(5): p. 80-93.

91. Kahn, K. and M.S. Miller, *Language Design and Open Systems*, in *The Ecology of Computation*, B.A. Huberman, Editor. 1988, North-Holland ; Sole distributors for the U.S.A. and Canada Elsevier Science Pub. Co.: Amsterdam ; New York, N.Y., U.S.A. p. 291-313.

92. Kiciman, E. and A. Fox. *Using dynamic mediation to integrate COTS entities in a ubiquitous computing environment*. in *Handheld and Ubiquitous Computing Second International Symposium HUC 2000*. 2000. Bristol, UK: Berlin, Germany : Springer-Verlag, 2000.

93. Killen, T., *Presentation to Industry Advisory Board*, . 2000, Stanford, CA., USA: Center for Integrated Facility Engineering, Stanford University. .

94. Kindberg, T., *et al. People, places, things: Web presence for the real world*. in *Third IEEE Workshop on Mobile Computing Systems and Applications*. 2000. Los Alamitos CA USA: Los Alamitos, CA, USA : IEEE Comput. Soc, 2000.

95. Kindberg, T. and A. Fox, *System Software for Ubiquitous Computing*, in *IEEE Pervasive Computing*. 2002. p. 70-81

96. Lamport, L., *Time, clocks, and the ordering of events in a distributed system.* Communications of the ACM, 1978. **21**(7): p. 558-65.

97. Langheinrich, M. *Privacy by Design - Principles of Privacy-aware Ubiquitous Systems*. in *Ubicomp 2001*. 2001. Atlanta, GA, USA.

98. Lea, R., *et al.*, *Networking home entertainment devices with HAVi.* Computer, 2000. **33**(9): p. 35-43.

99. Lehman, T.J., S.W. McLaughry, and P. Wyckoff. *T Spaces: the next wave*. in *HICSS 32 - 32nd Annual Hawaii International Conference on System Sciences*. 1999. Maui HI USA: Los Alamitos, CA, USA : IEEE Comput. Soc, 1999.

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

101. Liston, K., M. Fischer, and T. Winograd, *Focused Sharing of Information for Multi-disciplinary Decision Making by Project Teams.* ITcon, 2001. **6**: p. 69-81.

102. Liston, K., J. Kunz, and M. Fischer. *Requirements and Benefits of Interactive Information Workspaces in Construction*. in *8th International Conference on Computing in Civil and Building Engineering*. 2000. Stanford, CA, USA.

103. Malone, T.W., *What is Coordination Theory?*, Working Paper #2051-88. 1988, Cambridge, MA: MIT Sloan School of Management. .

104. Malone, T.W. and K. Crowston. *What is coordination theory and how can it help design cooperative work systems?* in *CSCW 90 Los Angeles. Proceedings of the Conference on Computer-Supported Cooperative Work*. 1990. Los Angeles CA USA: New York, NY, USA : ACM, 1990.

105. Mark, G., *Collaborative Design Within and Between Warrooms.* submitted to the Human-Computer Interaction Journal, 2002.

106. Mark, G., *Extreme Collaboration.* Communications of the ACM, 2002. **45**(4).

107. Martin, D.L., A.J. Cheyer, and D.B. Moran. *The Open Agent Architecture: a framework for building distributed software systems*. in *PAAM 98. Third International Conference on the Practical Application of Intelligent Agents and Multi-Agent Technology*. 1998. London, UK: USA : Taylor & Francis, 1999.

108. Microsoft Corporation., *Automation programmer's reference : using ActiveX technology to create programmable applications*. 1997, Redmond, Wash.: Microsoft® Press. xi, 477.

109. Miller, R. *Response Time in Man-Computer Conversational Transactions*. in *AFIPS Fall Joint Computer Conference*. 1968.

110. Munson, M., *System Support for Composing Distributed Applications Using Events*. Diploma Dissertation, Department of Computer Science. 1998, Cambridge, UK: Cambridge University.

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

112. Murphy, B. and T. Gent, *Measuring system and software reliability using an automated data collection process.* Quality and Reliability Engineering International, 1995. **11**(5): p. 341-53.

113. Myers, B., *Using Handhelds and PCs Together*, in *Communications of the ACM*. 2001. p. 34-41

114. NSF-IRIS, *A Report by the NSF-IRIS Review Panel for Research on Coordination Theory and Technology*. NSF Forms and Publications Unit Report, . 1989: NSF-IRIS. .

115. Oki, B., *et al.*, *The Information Bus-an architecture for extensible distributed systems.* Oper. Syst. Rev. (USA), Operating Systems Review, 1993. **27**(5): p. 58-68.

116. Panko, R.R., *Office work.* Office: Technology and People, 1984. **2**: p. 205-238.

117. Papadopoulos, G. and F. Arbab, *Coordination Models and Languages*, in *Advances in Computers: The Engineering of Large Systems*, M. Zelkowitz, Editor. 1998, Academic Press: New York, NY. p. 329-400.

118. Patterson, D. and A. Brown. *Recovery-Oriented Computing Keynote Address*. in *High Performance Transaction Systems (HPTS) Workshop*. 2001. Pacific Grove, CA.

119. Pham, T., G. Schneider, and S. Goose. *A Situated Computing Framework for Mobile and Ubiquitous Multimedia Access using Small Screen and Composite Devices*. in *ACM Multimedia 2000*. 2000. Los Angeles, CA: ACM, New York, NY, USA.

120. Picco, G.P., A.L. Murphy, and G.C. Roman. *Developing mobile computing applications with LIME*. in *International Conference on Software Engineering*. 2000. Limerick, Ireland: New York, NY, USA : ACM, 2000.

121. Picco, G.P., A.L. Murphy, and G.C. Roman. *LIME: Linda meets mobility*. in *21st International Conference on Software Engineering (ICSE '99)*. 1999. Los Angeles, CA, USA: New York, NY, USA : ACM, 1999.

122. Ponnekanti, S., *et al. ICrafter: A Service Framework for Ubiquitous Computing Environments*. in *Ubicomp 2001*. 2001. Atlanta, GA, USA.

123. Ponnekanti, S.R., L.A. Robles, and A. Fox. *User interfaces for network services: what, from where, and how*. in *Fourth IEEE Workshop on Mobile Computing Systems and Applications*. 2002. Callicoon, NY, USA: Los Alamitos, CA, USA : IEEE Comput. Soc, 2002.

124. Rekimoto, J. *Multiple-computer user interfaces: a cooperative environment consisting of multiple digital devices*. in *Cooperative Buildings Integrating Information, Organization, and Architecture First International Workshop CoBuild'98 Proceedings*. 1998. Darmstadt, Germany: Berlin, Germany : Springer-Verlag, 1998.

125. Rekimoto, J. *Pick-and-drop: a direct manipulation technique for multiple computer environments*. in *Tenth Annual Symposium on User Interface Software and Technology*. 1997. Banff Alta. Canada: New York, NY, USA : ACM, 1997.

126. Repenning, A. and J. Ambach. *Tactile programming: a unified manipulation paradigm supporting program comprehension, composition and sharing*. in *1996 IEEE Symposium on Visual Languages*. 1996. Boulder CO USA: Los Alamitos, CA, USA : IEEE Comput. Soc. Press, 1996.

127. Repenning, A. and T. Sumner, *Agentsheets: a medium for creating domain-oriented visual languages.* Computer, 1995. **28**(3): p. 17-25.

128. Roseman, M. and S. Greenberg, *Building Real-time Groupware with GroupKit, a Groupware Toolkit=.* ACM Transactions on Computer-Human Interaction (TOCHI), 1996. **3**(1): p. 66-106.

129. Rowstron, A. and A. Wood. *Solving the Linda multiple rd problem*. in *COORDINATION '96. First International Conference on Coordination Models and Languages*. 1996. Cesena, Italy: Berlin, Germany : Springer-Verlag, 1996.

130. Schuckmann, C., *et al. Designing object-oriented synchronous groupware in COAST*. in *Computer Supported Cooperative Work '96*. 1996. Cambridge, MA, USA: ACM.

131. Schuckmann, C., J. Schummer, and P. Seitz. *Modeling collaboration using shared objects*. in *GROUP 99: Conference on Supporting Group Work*. 1999. Phoenix AZ USA: New York, NY, USA : ACM, 1999.

132. Seymour, E., *Student Assessment of Learning Gains*, . 1997, University of Wisconsin http://www.wcer.wisc.edu/salgains/instructor/default.asp.

133. Spreitzer, M. and A. Begel. *More flexible data types*. in *IEEE 8th International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises (WET ICE'99)*. 1999. Stanford, CA, USA: Los Alamitos, CA, USA : IEEE Comput. Soc, 1999.

134. Stefik, M., *et al.*, *Beyond the chalkboard: computer support for collaboration and problem solving in meetings.* Communications of the ACM, 1987. **30**(1): p. 32-47.

135. Stone, M.C., *Color and brightness appearance issues in tiled displays.* IEEE Computer Graphics and Applications, 2001. **21**(5): p. 58-66.

136. Streitz, N., *et al. i-LAND: An interactive Landscape for Creativity and Innovation*. in *ACM Conference on Human Factors in Computing Systems (CHI'99)*. 1999. Pittsburgh, PA, USA: ACM Press, New York, NY, USA.

137. Streitz, N.A., *et al. DOLPHIN: integrated meeting support across local and remote desktop environments and liveboards*. in *Workshops of the Conference on Computer Supported Collaborative Work*. 1994. Chapel Hill NC USA: New York, NY, USA : ACM, 1994.

138. Streitz, N.A., J. Geissler, and T. Holmer. *Roomware for cooperative buildings: integrated design of architectural spaces and information spaces*. in *Cooperative Buildings Integrating Information, Organization, and Architecture First International Workshop CoBuild'98 Proceedings*. 1998. Darmstadt, Germany: Berlin, Germany : Springer-Verlag, 1998.

139. Tandler, P., *The BEACH Application Model and Software Framework for Synchronous Collaboration in Ubiquitous Computing Environments.* To appear in Journal of Systems and Software, 2003(Special Issue on Application Models and Programming Tools for Ubiquitous Computing).

140. Vahdat, A. and M. Dahlin. *Active names: flexible location and transport of wide-area resources*. in *USENIX'99: 2nd Symposium on Internet Technologies and Systems*. 1999. Boulder CO USA: Berkeley, CA, USA : USENIX Assoc, 1999.

141. Waldo, J., *The Jini architecture for network-centric computing.* Communications of the ACM, 1999. **42**(7): p. 76-82.

142. Weiser, M., *The computer for the 21st century.* Scientific American, 1991. **265**(3): p. 66-75.

143. Weiser, M., *Some computer science issues in ubiquitous computing.* Communications of the ACM, 1993. **36**(7): p. 74-84.

144. Winograd, T., *Interaction Spaces for 21st Century Computing*, in *HCI in the New Millennium*, J. Carroll, Editor. 2001, Addison Wesley.

145. Wyckoff, P., *et al.*, *T spaces.* IBM Systems Journal, 1998. **37**(3): p. 454-74.

146. Zhonghua, Y. and K. Duddy, *CORBA: a platform for distributed object computing. (A state-of-the-art report on OMG/CORBA).* Operating Systems Review, 1996. **30**(2): p. 4-31.