Extending Tuplespaces for Coordination in Interactive Workspaces

Brad Johanson and Armando Fox, Stanford University
Gates 3B-376, 353 Serra Mall
Stanford, CA 94305-9035

bjohanso@graphics.stanford.edu, fox@cs.stanford.edu

Abstract. The current interest in programming models and software infrastructures to support ubiquitous and environmental computing is heightened by the falling cost of hardware and the ubiquity of local-area wireless networking technologies. *Interactive workspaces* are technologically augmented team-project rooms that represent a specific sub-domain of ubiquitous computing. We argue both from related work and from our own experience with a prototype that the tuplespace model of communication forms the best basis for a coordination infrastructure for such workspaces. This paper presents the usage and characteristics expected of interactive workspaces, from which we derive a set of key system properties for any coordination infrastructure in an interactive workspace. We show that the design aspects of tuplespaces, augmented with some new extensions, yield a system model, which we call the Event Heap, that satisfies all of the desired properties. We also briefly discuss why other coordination models fall short of the desired properties, and describe our experience using our implementation of the Event Heap model. The paper focuses on a justification of the use of tuplespaces in interactive workspaces, and does not provide a detailed discussion of the Event Heap implementation or our more general experience with interactive workspaces, each of which is treated in detail elsewhere.

1 Introduction

Improvements in device technologies and falling costs enable ubiquitous computing, an approach proposed originally in [49]. Devices from large wall-sized displays to small PDAs can now be networked together in localized areas, wirelessly or otherwise, forming the hardware side of the ubiquitous computing environment. Once connected together, however, the problem is how to allow programs running on the devices to work together, either as architected or ad-hoc ensembles.

Our own project, Interactive Workspaces [28], investigates the systems and human-computer interface (HCI) issues that arise in technologically rich, room-based ubiquitous computing environments. Interactive workspaces are technologically-augmented team-project rooms that are used by groups to do collaborative problem solving. These spaces contain both permanent computational and I/O resources as well as portable devices that are brought in by participants and integrate with the environment. Compared to other room-based ubiquitous computing projects [5, 10, 12, 13, 18, 45], Interactive Workspaces features a stronger systems emphasis, focusing on providing software infrastructure for the dynamic interaction of heterogeneous and ad hoc collections of new and legacy devices, applications, and operating systems. Our experience with the "iRoom," our prototype interactive workspace, and the realities of these environments suggests that currently-used programming models are incomplete or inadequate.

During the past three years we have had the chance to go through several iterations of interactive workspace coordination infrastructure. Our experience the flaws and limitations of each generation informed the design of the next generation and led to a better understanding of the environment in general. Based on this experience, we can write down certain indispensable desiderata for our software infrastructure:

1. It must tolerate a dynamic environment: portable devices entering and leaving the room's wireless network should not disrupt other applications and devices.
2. It must enable the system (room) as a whole to maintain a high degree of robustness and availability despite inevitable (but usually transient) software and hardware failures. While this is generally true for all ubiquitous computing rooms, we are acutely aware of the problem as researchers since we work with rapidly prototyped, potentially instable, software and do not want experimentation to destabilize our existing system.
3. It must allow the rapid integration of new devices and systems. A common-language approach such as "Java everywhere" is not enough by itself: we wish to leverage devices and their existing application bases, i.e. Win32 productivity applications or Palm built-in PIM applications.
4. Both the software infrastructure and applications written to use it must be portable across installations, allowing as much as possible for the heterogeneity of specific equipment installed at each site. We would like a software infrastructure that can be to an interactive workspace what an operating system is to a PC—applications programmed for our infrastructure should run unmodified in any interactive workspace running our infrastructure.

We argue for a set of characteristics that must be taken into account by any software infrastructure for an interactive workspace (Section 3). These characteristics in turn imply certain properties for software infrastructures in this domain (Section 4); we show that tuplespaces, a referentially and temporally decoupled coordination model described in more detail later, have a set of features that give them most of the desired software infrastructure properties, and introduce some extensions to provide the properties tuplespaces lack (Section 5). Specifically, these extensions are: self-description, flexible typing, standard routing fields, tuple expiration, query registration, ordering, and modular restartability.

We refer to tuplespaces plus this set of extensions as the Event Heap, which is also the name of our prototype implementation. In large part the extensions are needed to overcome the transition from using tuplespaces as a closed system, in which all participating processes are designed to work together, to an open system, in which a more diverse collection of applications needs to interoperate. Note also, that while we could have chosen a different coordination model as our point of departure, tuplespaces required the fewest changes for the interactive to be compatible with the interactive workspace s environment. This paper presents our arguments to support this claim.

In addition, although it has previously been possible to specify what a technology rich team-project space might look like, there has been no standard concept of how to enable an application for use in such a space. The Event Heap model advocated in this paper provides such a concept.

This article gives systems arguments for the utility of tuplespaces and the need for the given extensions in the interactive workspace domain, but provides few details of our prototype implementation of the Event Heap and only a brief discussion of our extensive experience with it; the interested reader may see [27]. Further, although the Interactive Workspaces project is quite broad and, in particular, also has a strong focus on Human Computer Interaction (HCI) problems for this domain, we do not go beyond system arguments as to the utility of tuplespaces; a good overview of the project is provided in [28].

2 An Example Scenario

Although most of this paper will argue in terms of abstract properties, we feel that a concrete interactive workspace usage scenario will help to motivate those arguments. We provide here a scenario set in the iRoom that reflects how we feel such spaces may be used. The scenario is set in the iRoom, and is based on research we have done with the Center for Integrated Facilities Engineering (CIFE) [33], a civil engineering research group on campus. The iRoom looks like a standard conference room with technological additions. Specifically, there are three six foot diagonal touch sensitive displays along one wall, a bottom projected table, a 6 foot diagonal high resolution front display, and wireless support for PDAs and laptops that are brought into the space. All the displays are driven by one or more stand alone PCs. A more detailed video version of this scenario may be found on-line at: http://iwork.stanford.edu.

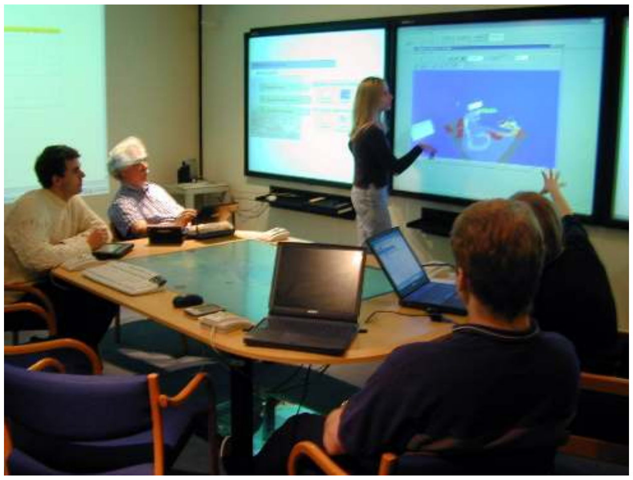


Figure 1 - Construction Management in the iRoom

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 1 shows a mockup of the iRoom being used for such a scenario.

Later in the meeting, the leader is suddenly called away and turns off his laptop which 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 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 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 3D model 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 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 [14, 26, 34, 35] 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 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 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. Table 1 provides a summary of interactive workspace characteristics, and the remainder of this section discusses them in more detail.

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 1- Summary of Interactive Workspace Characteristics

3.1 The Human Side

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

Bounded Environment

An interactive workspace is bounded in physical extent, therefore humans expect devices and applications to coordinate with one another within the space. By the same token, coordination with applications and devices outside of the space should not occur unless a user specifically requests the coordination. In the scenario from Section 2, for example, the construction engineers expect their laptops to interact with the room software 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). One of the authors has previously identified this as the 'boundary principle' [31]. 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. The bounded nature of an interactive workspace we refer to as **H1**.

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 [34, 35]. 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. In other words, the coordination system needs to support flexible reconfiguration and dynamic coordination.

From the systems perspective, the software environment of an interactive workspace will consist of ensembles of applications that dynamically coordinate with one another. This style of interaction is evident in the scenario we lay out in Section 2 where the construction engineers pull up various applications to better understand various problems. 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**.

Human Performance Constraints

Since workspaces will be used by humans, there is 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 [26]. This observation can be used to establish a limit on the number of devices and amount traffic a coordination 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 new lights brought into the room shouldn't turn themselves on.

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

3.2 The Technology Side

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.

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 in Section 2, 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 [23, 25]. This type of hardware heterogeneity we refer to as characteristic **T1**.

Similar arguments apply to the software components in 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 Section 2, for example, rely on a custom developed 4D modeler that displays a 3D model of construction state for any specific date during construction); it is usually impractical or impossible to rewrite the applications just to make them work in an interactive workspace. Further, even when software is custom-written, hardware heterogeneity suggests that any coordination infrastructure should work with as many software platforms as possible. We refer to the variety of software environments that will be present in an interactive workspace as characteristic **T2**.

Changing Environment

An 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 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 Section 2, 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. 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**.

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

4 Properties for Coordination Infrastructures

The characteristics of interactive workspaces discussed in the previous section lead to a set of properties that a coordination infrastructure designed to support user collaboration in such a space needs to support. Table 2 provides a summary of the properties and how they relate to the characteristics discussed in Section 3, while the rest of the section describes the properties and their relationships to interactive workspace characteristics 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 which can be composed. • *Long Timescale Change* (**T4**) by providing a broader selection of new applications with which to evolve workspace functionality.

Table 2 - Necessary Coordination System Properties

4.1 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 before they became connected (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 topic" message might be relevant for several minutes). 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 what to do with 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.

4.2 Referential Decoupling (P2)

Referential decoupling allows entities to communicate with one another without naming each other directly. 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 using a level of indirection such that senders and receivers only interact through an intermediary (for example, in the Metaglue system [13]). 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 [6]).

Referential decoupling makes it harder to create applications that are tightly interdependent since applications are not programmed to interact with pre-specified peers according to some pre-defined pattern. This provides for the short-term changes in an interactive workspace (**T3**) since applications are less likely to crash as a result of the disappearance or failure of another application with which they were 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 which is an aspect of human-centered interaction (**H2**) in an interactive workspace.

4.3 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 without modifying existing applications, or having access to their source code. This allows integration of task-specific legacy applications for use in an interactive workspace environment, supporting **H2**.

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

- **Snooping:** This allows applications to spy on 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 the pre-existing set. For example, a smart-classroom which allows the professor to coordinate several electronic whiteboards could have an application added to it that displays the whiteboard state on students laptops.

- **Interposability:** This technique allows an intermediate 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 speak different message protocols to interact with one another without having to modify either programs code.

- **Stream Transformation:** This technique is related to the other two but is more general. Stream transformation 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) [9].

4.4 Expressiveness (P4)

The 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 hook 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 need 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.

4.5 Simple and Portable Client API (P5)

The number of API calls supported, and the amount of code to support the software infrastructure in the per application client code should be minimized. The main reason for this is to simplify the task of making new platforms compatible with the coordination system by making it easy to port 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 this requirement is seemingly in contradiction with **P4**, which advocates an API that allows many different coordination patterns to be expressed. As will be discussed in more detail in the sub-section of 5.1 on Infrastructure API, just as it is possible 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 which are sufficient to express a variety of coordination types.

4.6 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 in such a way as to make it easy to figure out and debug interactions. This also makes it easier to trouble shoot the integration of new devices and applications into the workspace, and thus supports workspace evolution (**T4**).

4.7 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 10 ms to about 1 s of latency depending on the type of action taken by the human [8]. More specifically, Miller [36] 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 properties of interactive workspaces.

4.8 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 of devices, and that humans will specifically interact with applications during meetings on the order of one time per minute. 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 think there will be on the order of tens of events per minute that the system will need to be able to handle with a latency of less than 100 ms.

4.9 Failure Tolerance and Recovery (P9)

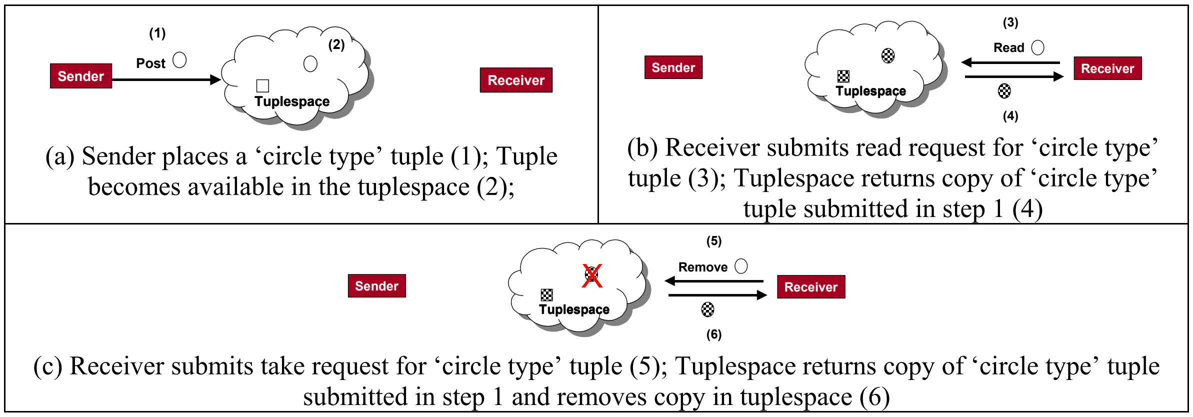
In order to be productive, users must not be continually interrupted during collaboration. This means that failures in components should not cause other components or the software infrastructure to fail. Individual applications may be failure prone, either due to rushed development time on commercial products or due to buggy research applications, and, although the coordination infrastructure system should be constructed to be robust, it too may fail on occasion. The system therefore needs to provide mechanisms for coping with these failures in order to allow for the short term dynamism (**T3**) of the workspace. The same mechanisms can support human-centered interaction (**H2**) by minimizing disruptions to collaborators in an interactive workspace.

4.10 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—there should be nothing about the coordination infrastructure that specifically associates 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.

5 Tuplespace Features and Needed Extensions

Tuplespaces were first proposed by Gelernter and Carriero as a coordination system for parallel programs. They proposed the Linda [7] *tuplespace* model. A tuple is a set of ordered typed fields, each of which either contains a value or is undefined; a tuplespace is an abstract space containing all tuples and visible to all processes (the original Linda system was designed for coordination of processes for parallel computing, so we use "processes" here.—in our system and other modern tuplespace implementations, however, full applications are also allowed to participate). The most important language primitives 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 match template that may specify explicit values or wildcards for any tuple fields. Figure 2 shows an abstract representation of how tuplespaces function.



(a) Sender places a 'circle type' tuple (1); Tuple becomes available in the tuplespace (2);

(b) Receiver submits read request for 'circle type' tuple (3); Tuplespace returns copy of 'circle type' tuple submitted in step 1 (4)

(c) Receiver submits take request for 'circle type' tuple (5); Tuplespace returns copy of 'circle type' tuple submitted in step 1 and removes copy in tuplespace (6)

Figure 2 - Tuplespace System Diagram (shaded objects are tuples, and hollow objects are templates)

In this section we present the design aspects of the tuplespace system model that give it some but not all of the properties that were advocated in Section 4, followed by the extensions needed to satisfy the remaining properties. Table 3 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 described in this section provide for all of the previously defined properties.

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.
Persistence	
F4. Limited Data Persistence*	• *Limited Temporal Decoupling* (**P1**). • *Failure Tolerance and Recovery* (**P9**) by preventing tuple buildup which 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 more easily be 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 3 - System Features and Related Properties

[*] Not provided by basic tuplespace model.

A variety of tuplespaces have been introduced since Linda which modify the basic features, in some cases making them more compatible with the needs of interactive workspaces. Nonetheless there is no canonical replacement to the original Linda tuplespaces model, so we use that for comparison. Section 5.4 summarizes some of the more prominent "modern" tuplespace systems and how their features align with the extensions we propose to the basic tuplespace model.

5.1 Design Aspects 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, or 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 explicitly, or logically through a level of indirection. Tuplespaces provide logical routing through the use of content based addressing, since 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 [6].

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 receiver match for a specific value in a tuple field indicating it is the recipient and 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.

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. Unfortunately, we identified *limited* temporal decoupling (therefore limited persistence) as the desired property, 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 5.2. 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 specifying 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, which makes 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 is discussed in Section 5.2.

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 matching events until the query is removed.

Transparency vs. Opacity of Communication

Transparency of communication is the degree to which applications using the coordination 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, however, determine which applications receive copies of any given tuple). *Transparent communication* we refer to as feature **F6**. The transparency provided makes the system more extensible (**P3**) by allowing snooping and interposability (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 address later in Section 5.2 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, 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 described in Section 5.2 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 makes the latter likely, but without sophisticated artificial intelligence the former would remain unlikely.

Distribution of Infrastructure

Another design decision is to what extent to the software 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 coordination 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 [3] and T Spaces [50], 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 reduces performance and limits 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 used by a group collaborating 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 [22] 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 date 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 both by **P7** and **P8**.

Infrastructure API

The tuplespace model has a very simple API (**P5)** with six functions that are used to express the coordination (the three mentioned at the beginning of Section 5, 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 easy to add wrappers to existing application's programmatic interfaces when source code is unavailable (similar to "puppeteering" [16]), 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 turns can be made quite simple. The *simple API* feature we call **F11**.

In the Event Heap we have C++, Java, and Python native support, and using COM and OLE under Windows we've been able to support Visual Basic and integrate with Office applications. We also have a web interface for submission of events through a Java servlet. We attribute in large part our ability to support so many different interfaces to the simplicity of the API. Using the various platforms has also enabled us to integrate with legacy applications in a fairly straightforward manner. For example, we were able to integrate Office applications into our system in little more than the time to learn the COM interface using our Java code and a Java-COM bridge library. Integrating with CIFE's 4D viewer (one of the applications in the scenario from Section 2) took a little less than 100 lines of commented C++ code added into their original application.

Another important design characteristic of the API is its generality. Gelernter and Carriero [21] 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 procedural language being Turing-complete).

A main feature of the tuplespace model, they argue, is that it is in fact 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 calling application and two on the application receiving the call. This generality provides for portability to new platforms and 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.

5.2 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 stem from its original intent as a coordination system for parallel applications, in which all the processes were designed together. This means that there are never any issues of coordination among processes that are foreign to one another.

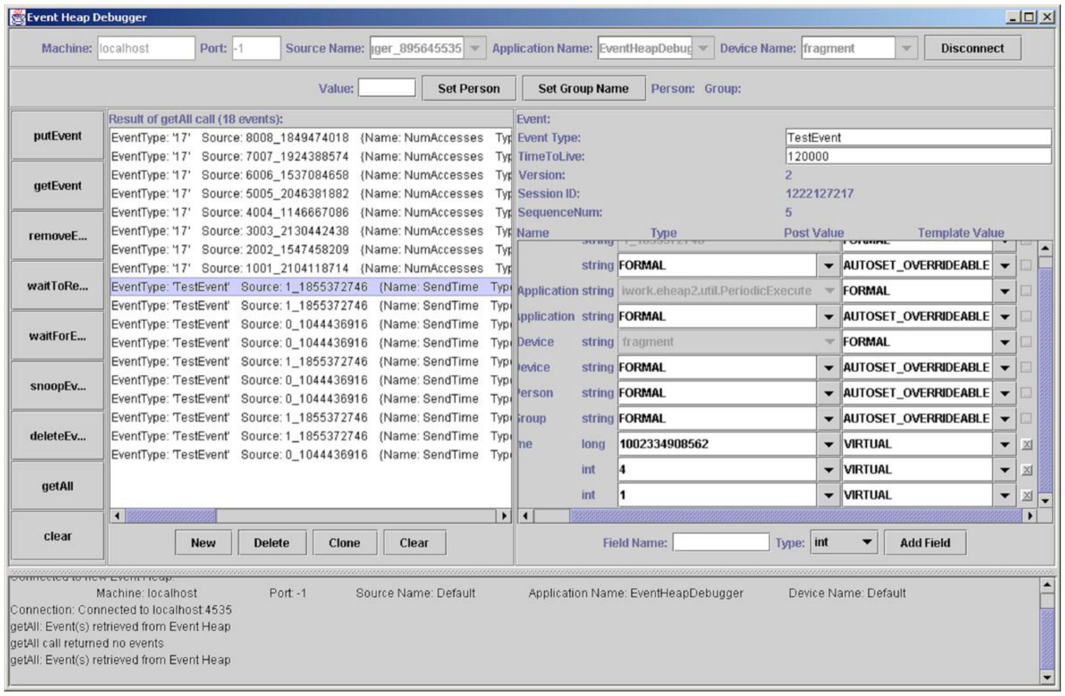
The remainder of this section details a set of extensions to the tuplespace model to address its deficits in the interactive workspace domain. We call tuplespaces plus these extensions the Event Heap model. The model has been implemented in a system with the same name [27]. Tuples used by the Event Heap are slightly different from those in the basic tuplespace model, being typed and having other standard fields, among other things. To distinguish them from basic Linda-style tuples, we refer to them as events throughout the remainder of the paper. Other extended tuplespaces systems that have implemented some of these features are discussed in Section 5.4.

Self-describing Tuples

One problem with using tuplespaces in an interactive workspace is that the semantic meaning of fields in a tuple is only known by the programmer creating the processes that use the tuplespace. This makes it difficult to reverse engineer tuple communication to integrate new applications with an existing collection of cooperating applications. To get around this problem, the tuple fields can be made self-describing by adding a string containing a name to the type and value for each field. We call this feature **F7**[1]. With self-description, a field can, for example, be called 'Xpos' if it contains the x position for the data being represented by the tuple. Combined with tuple typing, the extension which 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). This provides for additional extensibility (**P3**) and improves application portability (**P10**). Finally, it also makes it easier to debug the system by monitoring traffic in the

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

tuplespace (**P6**). Self-describing fields have been used in other extended tuplespaces implementations including TSpaces [50].



Machine: localhost Port: -1 Source Name: ger_895645535 Application Name: EventHeapDebug Device Name: fragment

Value: Person: Group:

Result of getAll call (18 events):

```
EventType: '17'  Source: 8008_1849474018  {Name: NumAccesses  Typ
EventType: '17'  Source: 7007_1924388574  {Name: NumAccesses  Typ
EventType: '17'  Source: 6006_1537084658  {Name: NumAccesses  Typ
EventType: '17'  Source: 5005_2046381882  {Name: NumAccesses  Typ
EventType: '17'  Source: 4004_1146667086  {Name: NumAccesses  Typ
EventType: '17'  Source: 3003_2130442438  {Name: NumAccesses  Typ
EventType: '17'  Source: 2002_1547458209  {Name: NumAccesses  Typ
EventType: '17'  Source: 1001_2104118714  {Name: NumAccesses  Typ
EventType: 'TestEvent'  Source: 1_1855372746  {Name: SendTime  Typ
EventType: 'TestEvent'  Source: 1_1855372746  {Name: SendTime  Typ
EventType: 'TestEvent'  Source: 0_1044436916  {Name: SendTime  Typ
EventType: 'TestEvent'  Source: 0_1044436916  {Name: SendTime  Typ
EventType: 'TestEvent'  Source: 1_1855372746  {Name: SendTime  Typ
EventType: 'TestEvent'  Source: 0_1044436916  {Name: SendTime  Typ
EventType: 'TestEvent'  Source: 0_1044436916  {Name: SendTime  Typ
EventType: 'TestEvent'  Source: 1_1855372746  {Name: SendTime  Typ
EventType: 'TestEvent'  Source: 1_1855372746  {Name: SendTime  Typ
EventType: 'TestEvent'  Source: 0_1044436916  {Name: SendTime  Typ
```

Event:
Event Type: TestEvent
TimeToLive: 120000
Version: 2
Session ID: 1222127217
SequenceNum: 5

Name	Type	Post Value	Template Value
	string	FORMAL	AUTOSET_OVERRIDEABLE
Application	string	iwork.eheap2.util.PeriodicExecute	FORMAL
pplication	string	FORMAL	AUTOSET_OVERRIDEABLE
Device	string	fragment	FORMAL
evice	string	FORMAL	AUTOSET_OVERRIDEABLE
erson	string	FORMAL	AUTOSET_OVERRIDEABLE
roup	string	FORMAL	AUTOSET_OVERRIDEABLE
ne	long	1002334908562	VIRTUAL
	int	4	VIRTUAL
	int	1	VIRTUAL

Field Name: Type: int

```
        Machine: localhost        Port: -1        Source Name: Default        Application Name: EventHeapDebugger        Device Name: Default
Connection: Connected to localhost:4535
getAll: Event(s) retrieved from Event Heap
getAll call returned no events
getAll: Event(s) retrieved from Event Heap
```

Figure 3 - The Event Heap Debugger

So far in our project, self describing fields have proven most useful in debugging and adapting applications. We have written a debugger application (Figure 3) 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 and then testing out how changing field values and resubmitting events effect the results.

Flexible Typing

In a parallel application, the programmer or programmers can determine ahead of time standard formats and meanings for the tuples that will be exchanged. 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**[2]. Figure 4 shows some of the problems that can arise by comparing matching in a basic tuplespace extended only with self-description, and 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 field. Now applications need only avoid collisions on the name of this type, which solves the problem for the majority of the cases and provides for application portability to new spaces (**P10**). The

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

type of problem fixed by this is shown in Figure 4a. Typing has been implemented in other extended tuplespace systems such as JavaSpaces [3], and L2imbo [15].

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 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. The issue of typing and naming is further discussed in Section 8.2.

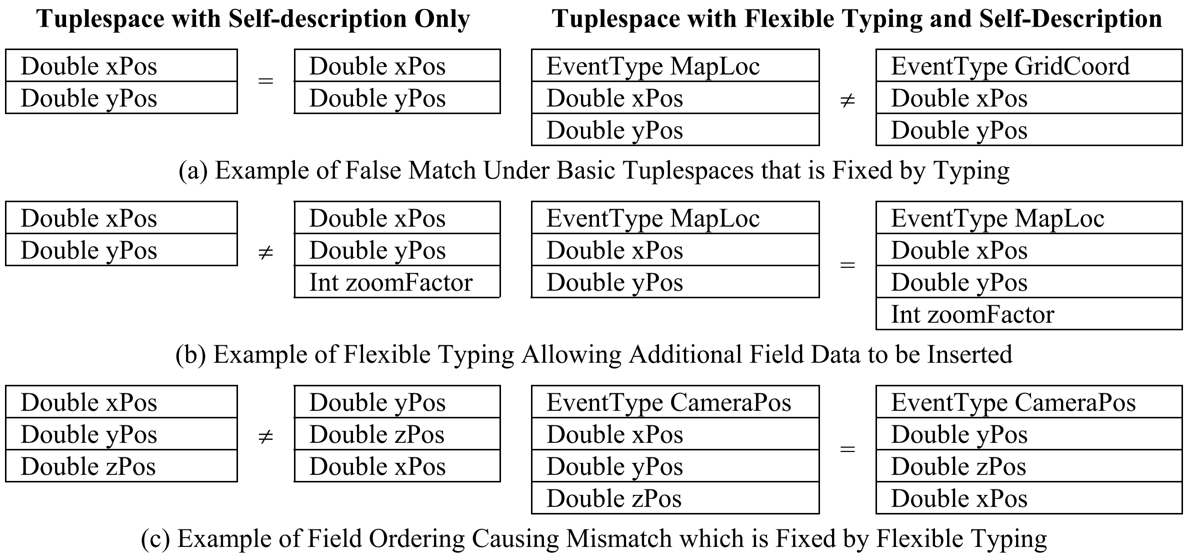


Figure 4 - Flexible Typing Examples ('=' : Tuples will match if values match; '≠' : Tuples can not match)

In addition, there is a problem of allowing newer applications to use an enhanced version of a tuple while maintaining interoperability with older applications. To get around this, the typing of tuples can be made 'flexible.' Specifically, matching can be changed to ignore field order and allow matching to any tuple that has a super-set of the template's fields (as in Figure 4b). Thus, newer applications can add fields with supplemental information without breaking compatibility, analogous to adding experimental headers in HTTP, making the system extensible (**P3**). Flexible typing has been advocated for software systems in general in [44]. 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.

We have needed the flexible typing feature on several occasions during development of applications for our project. The multibrowsing system [30] 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

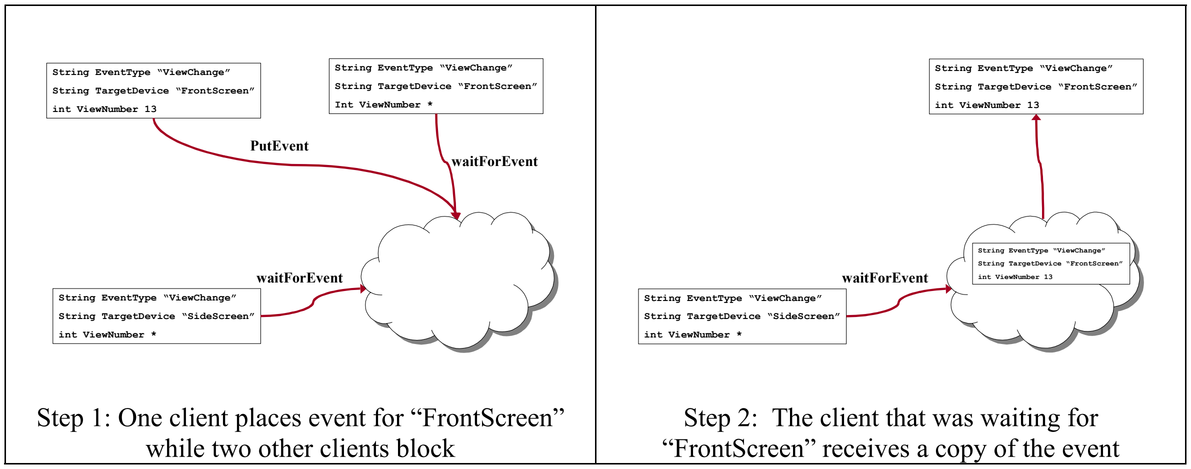
While the standard tuplespace model supports all of the routing patterns (one-to-one, broadcast, etc.), individual application developers must have a convention for which tuple fields are set to determine their routing. This works well for parallel applications with processes designed to work together. 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 application portability (**P10**). We call the addition of standard routing fields feature **F3**[3].

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

As an example, to send to all '3DViewer' applications, a sender can set the 'TargetApplication' field on a tuple to '3DViewer.' Since the client-side of the coordination infrastructure sets the 'TargetApplication' field of template tuples on all 3D Viewer applications to '3DViewer,' the tuple will only be picked up by 3D Viewer applications (but only by those that would otherwise match the tuple which was sent). If a tuple is sent without the application writer overriding the value of 'TargetApplication' field, that field will be sent out set to a wildcard value and will therefore match all receivers looking for a tuple 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 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 area. Figure 5 gives another example of using the routing fields to either send to a specific screen, or multicast to all screens.

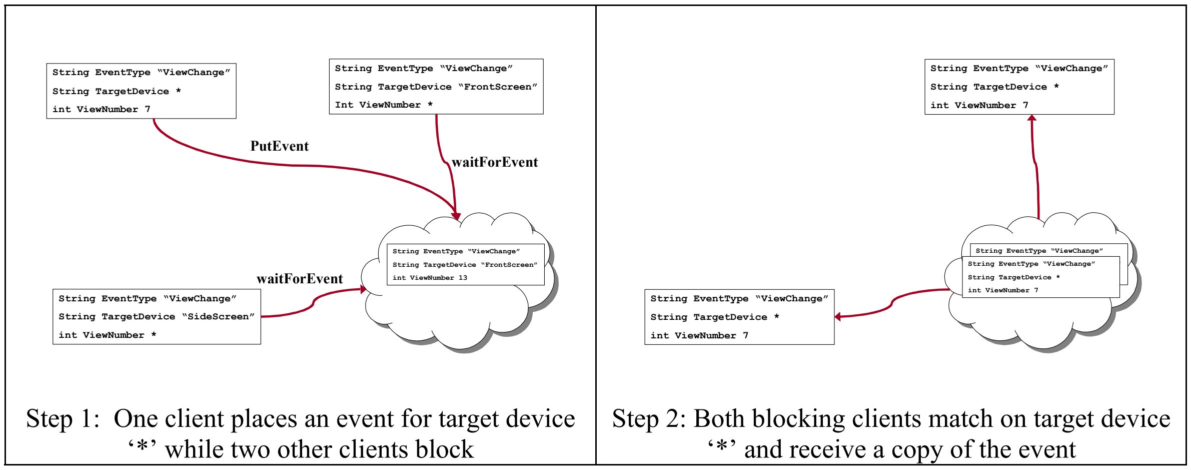
[3] This features number appears out of order, but is actually correct since it falls in the category of routing along with content-based addressing and support for all routing patterns (**F1** and **F2**, respectively).


String EventType "ViewChange"
String TargetDevice "FrontScreen"
int ViewNumber 13
PutEvent
String EventType "ViewChange"
String TargetDevice "FrontScreen"
Int ViewNumber *
waitForEvent
waitForEvent
String EventType "ViewChange"
String TargetDevice "SideScreen"
int ViewNumber *

Step 1: One client places 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".


String EventType "ViewChange"
String TargetDevice *
int ViewNumber 7
PutEvent
String EventType "ViewChange"
String TargetDevice "FrontScreen"
Int ViewNumber *
waitForEvent
waitForEvent
String EventType "ViewChange"
String TargetDevice "SideScreen"
int ViewNumber *

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 5 - Using the Event Heap Standard Routing Fields

We do not have a lot of experience yet 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 (**P9**). Extending tuplespaces by adding an expiration field to all tuples which is set by application developers gives the system limited temporal decoupling (**P1**), and 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 provide the limited data persistence feature (**F4**). Other extended tuplespace implementations, including TSpaces [50] have expiration for similar reasons. While other systems have used garbage collection for tuple expiration, the Event Heap uses a single thread that deletes tuples as they expire.

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 avoided. 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 which 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 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 makes 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) to spy on and react to communications even between applications which 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**[4]. Query registration has been a popular feature in other extended tuplespace implementations including JavaSpaces [3],TSpaces [50] and Lime [38].

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 of tuples returned by a series of identical queries, which 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, the application must track which tuples it has seen, which adds to development time. This problem of duplicate retrieval is known as the multiple read problem, and has been noted by [43]. This additional development time could discourage developers from doing it the right way and lead them to use destructive reads instead (in fact we saw this in early Event Heap versions that didn't support ordering).

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

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

add per-source FIFO, at most once, ordering or better 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. Since most applications do not need total ordering, only per source FIFO, at most once ordering is specified. The Event Heap implementation does in fact provide total ordering in most cases as a side effect of having a physically centralized server based implementation. Per-source ordering is also in LIME [38] and has also been found useful when applying the tuplespace model to other domains. See, for example, [32]. 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 switch to displaying different video sources. Unfortunately, using basic tuplespace semantics through TSpaces (which we were then using as the underlying system for the Event Heap), the same projector event was picked up over and over again 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 coordination infrastructure.

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**). Any application or even a component of the coordination 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**.

5.3 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.

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 systems 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 our measurements and an analysis of more efficient implementations of the model are left for a future publication.

5.4 Comparison to Modern Tuplespace Systems

As has been mentioned throughout the previous section, some of the extensions we found to be necessary have been included in other systems based on the tuplespace model. While none of these modern tuplespace systems have all of the features we find to be necessary, all of them could potentially be extended to make a suitable coordination infrastructure for interactive workspaces.

Table 4 summarizes where several of the modern tuplespace systems fall with regards to the extensions we have just proposed. 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). To avoid this problem in the Event Heap we have a standard TCP/IP based wire protocol and simple event specification that are language independent.

The four tuplespace-based systems that appear in the table are Gigaspaces [1], TSpaces [50], L2imbo [15] and LIME [38]. Gigaspaces is the first enterprise level commercial product that implements and extends the basic JavaSpaces [3] specification proposed by Sun. TSpaces is an IBM project, and is designed to be a general tool for ubiquitous computing type applications. L2imbo seeks to use filtering of events between multiple tuplespaces as a mechanism for providing Quality of Service (QoS) guarantees. LIME stands for Linda in a Mobile Environment and was developed to explore how tuplespaces can be made to function in an environment where connectivity between devices is constantly changing as users with mobile devices roam around.

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 [38, 40, 41].
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.
11. 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 4 - Tuplespace Extension Support in Modern Tuplespaces (✓ = has feature, × = doesn't have feature, ~ = limited support of feature)

As can be seen, none of the systems have 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.

6 Other Alternatives

While in the end we determined tuplespaces to be the coordination infrastructure best suited to interactive workspaces, there are many other coordination infrastructure that were candidates. Table 5 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).

	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 5 – Comparison of How Well Desired Interactive Workspace System Properties are Met by Various Coordination Infrastructures (✓ = fully meets property, × = doesn't have property, or limited, ~ = mostly meets property)

The remainder of the section describes in more detail how RMI/RPC, Publish-Subscribe and Message-Oriented-Middleware (MOM) fall short of the desired properties. 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 total 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).

6.1 RMI/RPC Systems with Rendezvous

Both RPC (remote procedure call) and RMI (remote method invocation, RPC's object-oriented equivalent) make execution of a procedure or method on a remote process appear local. Since they mimic a function call, all communication is transient and synchronous, with the calling application blocking until the result is returned. Calls cannot work until an appropriate remote target is found, so a proper coordination infrastructure must provide some mechanism for applications to rendezvous, or find out about one another. Through that mechanism the caller obtains the handle to use for the procedure call or method invocation.

[*] 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.

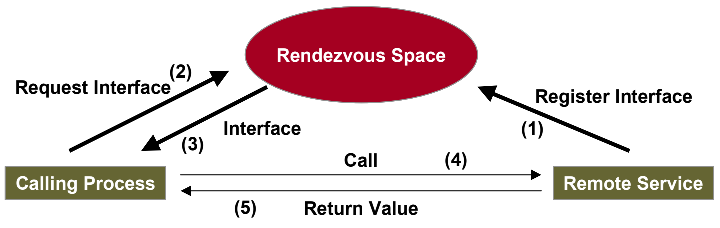
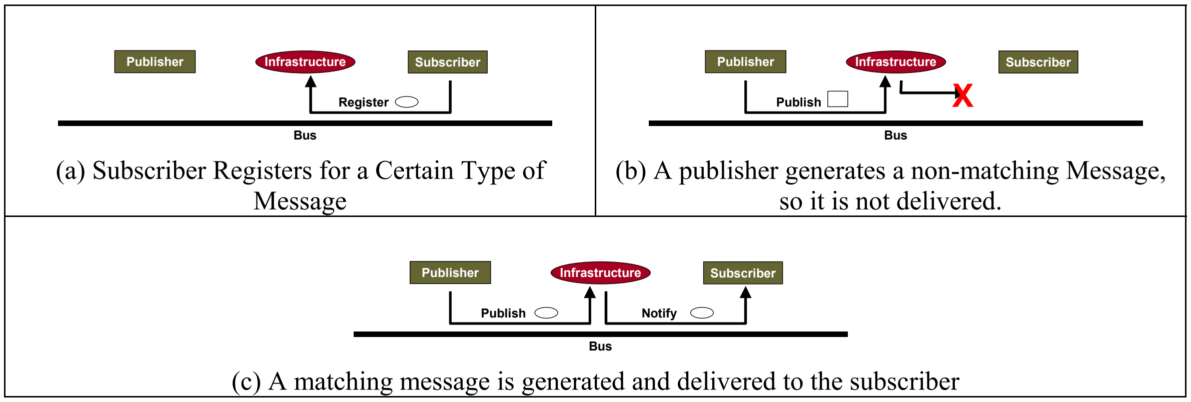


Figure 6 - RMI-Rendezvous System Diagram: Remote services register interfaces (1); Calling processes retrieve interfaces (2,3) and then make invocations on the remote service (4,5)

As an interactive workspace coordination infrastructure, RMI/RPC has many drawbacks. Since the system is synchronous and transient, it is fundamentally coupled both referentially and temporally. This tight coupling also means applications are strongly 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. It is relatively hard to extend collections of applications using RMI/RPC 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. The system can't perform most routing types without client applications implementing most of the logic themselves (broadcast, for example) so it is relatively non-expressive. APIs are typically not very 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. Finally, since all communication is direct and opaque it can be difficult to debug collections of applications using RMI/RPC. CORBA [51] and Jini [48] are examples of this type of coordination model, although to be fair both include extensions that allow them to avoid some of the drawbacks of a basic RMI/RPC system with rendezvous. An example of a RMI-Rendezvous system is shown in Figure 6.

6.2 Publish-Subscribe Systems



(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 7 - Publish-Subscribe System Diagram

Publish-Subscribe systems work entirely using a mechanism similar to the query persistence/registration extension for tuplespaces used by the Event Heap. Applications subscribe to messages of interest, and then receive a copy anytime another entity publishes a matching message. The basic mechanism is shown in Figure 7. Messages of interest can be specified by a channel, a subject, or more complex content based matching schemes similar to those used by tuplespaces. Perhaps the best example of a publish-subscribe system is the InfoBus [39]. Eugster et al. give a good overview of publish subscribe systems and their properties [19]. Also, the SIENA system has shown how publish-subscribe can be extended to Internet-scales while keeping performance reasonable [11].

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).

6.3 Message-Oriented-Middleware (MOM)

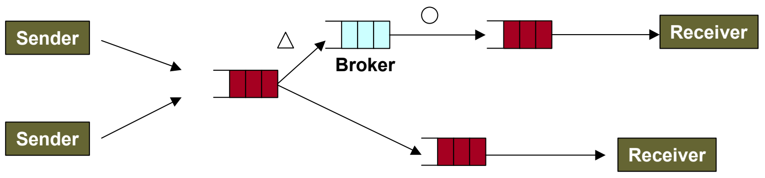


Figure 8 - Message Oriented Middleware System Diagram: Senders emit objects which are passed through several queues, eventually arriving at the appropriate location or locations

Message-Oriented-Middleware (MOM) is a less specific type of coordination model. It is sometimes referred to as message-queuing, and often includes features similar to publish-subscribe. MOM systems are intended to route messages between computer systems at sub-organizations within a corporation, or between computer systems within one or more corporations (for example routing of money transfers). Their main intent is to guarantee that request messages are delivered and that result messages are returned, so they usually provide transactions, fault tolerance and guaranteed delivery. Figure 8 shows an abstract representation of a MOM system.

In one style of MOM system, senders generate a stream of messages with a given target. A series of one or more intermediate servers routes the messages to the target. If the target is down, the messages are buffered for as long as needed. This decouples the applications in time, so they need not be running simultaneously. This style of operation is similar to IBM's MQ-Series [4], for example.

Other MOM systems allow content such as stock quotes to be sent into the coordination infrastructure, where intermediate servers and applications may consume them, transform their formats, or combine them with other messages before re-publishing them. Receivers specify content they want to receive, and the system collects and queues the information for them, even if they go down. This latter style of operation is similar to Gryphon [2].

Since MOM systems vary quite a bit, it is hard to specify how they compare on the properties. 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. Also, since many are designed to provide transactions or work over Internet scales, the latency in the system cannot be guaranteed to be perceptually instantaneous. 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).

7 Discussion

7.1 Experience

Our prototype iRoom includes three SmartBoard wall-size touch-sensitive displays, a hi-res Mural display, a tabletop display, wireless LAN and wireless pointing devices, and integration with laptops and PDA's. Space does not allow a detailed description of our extensive experience with the iRoom to date. We routinely use several applications that rely on the functionality of the Event Heap, including PointRight [29] for multi-display mouse control, multibrowsing [30] for web browsing across displays, ICrafter [42] for controlling lighting and projectors from handheld and other devices [20], SmartPresenter for multi-display presentations, and iStuff wireless buttons that can be programmed as "macros" (e.g. power up the entire room). Most of these were realized by combining a diverse array of off-the-shelf applications (e.g. PowerPoint) with tens or hundreds of lines of 'glue' code. As a research lab, the room has been remarkably robust under day-to-day use, and also supports several outside groups that use it for demos and as a facility for their own (non-Computer Science) projects. The system is also now deployed in several other interactive workspaces on Stanford campus, and is also being used at other locations around the world. We cannot prove that the combination of properties we argue to be key for coordination in an interactive workspace are complete, but we believe our implementation of a model with those properties, and its subsequent deployment in multiple locations, has demonstrated that they are a natural fit for such environments.

7.2 Applicability of Model

As discussed, this paper only argues for the suitability of an extended version of tuplespaces 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 bounded region of human and device interaction. It could be used for coordination among 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.

Many of the features of the model that provide for interaction of diverse devices, extensibility and the other mentioned properties come at the expense of scalability and performance. The system model is able to sacrifice in those regions due to the bounded scope of coordination and the reduced need for low latency that stem from the system being designed for a small group of interacting humans. Nonetheless, these limit the applicability of the system to other domains.

The limited scalability that comes from having a logically centralized mechanism of information exchange means that the model is not suitable for coordination across tens or hundreds of thousands of machines across the entire Internet. Nor is it suitable for coordination of large widely distributed sensor networks.

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 adds overhead and is not needed when the collection of processes are designed to work together. So, our new model would be inappropriate for high performance parallel computing applications even within an interactive workspace.

There are some types of communication among devices and applications in an interactive workspace 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 which 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.

8 Open Issues

While we have identified tuplespaces as a good starting point as a coordination infrastructure for an interactive workspace, and have specified a set of extensions that can be added to create a complete model for coordination, there are nonetheless several open issues that we have yet to fully address.

8.1 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**).

On the other hand, one of the design features of tuplespaces and the Event Heap is the transparency of data and communication. Together they make it easier to debug collections of applications and make the system more extensible. These same features, however, make the system insecure and public to all users in an interactive workspace. Unfortunately, it is not obvious how to fix this. While there are many obvious security fixes that could be made—for example, allow tuples to be partially encrypted—many of them would degrade the ability to debug and extend the system, and it is not obvious whether this is what is desired.

The main problem is that a social model for security in an interactive workspace needs to be determined. Unlike communication across the Internet, coordination within the workspace occurs due to interactions of individuals with applications within the space. Just as there is an implicit agreement when people meet in a room that all conversations there are public, it seems that when people collaborate within an interactive workspace there should be an implicit agreement that all "electronic conversations" are also public. On the other hand, sometimes charts or photos are shared at a meeting with the assumption that no record is taken out with participants. When that same information is electronic it can easily be captured and used at a later time.

For the time being our strategy has been to firewall the interactive workspace so that no outside machines have access to the traffic in the room. We realize that a more detailed analysis of the social and technological issues related to security in interactive workspaces is needed.

8.2 Type Collisions

The issue of type collision arises in any system that allows interoperation of applications not designed to work together, so it is of course an issue for any coordination infrastructure designed to work in an interactive workspace. There are two main issues: name collision on tuple types and use of different tuple type names for similar information. Type collision occurs when two application designers choose the same tuple type name for tuples with different content. We expect that this will be relatively rare since we use strings for type names. Further, since field names and types must also match for the tuples 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 systems can be found earlier in this paper in the flexible-typing part of Section 5.2 .

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 tuples types that control their projectors. Fortunately, adapters can be made using the interposability feature of the system. An adapter is a software intermediary that can pick up tuples of one type and convert them into tuples of another type which then get deposited in the tuplespace. Other researchers have done work on this in the past, for example with the Event Exchange system [37] which worked with T Spaces [50]. We have an initial version of an application called the "Patch Panel" that allows these type conversions to be easily set up either programmatically through sending events to a well known Event Heap service, or through an easy to use GUI.

8.3 Improving Performance and Integrating More Communication Types

As discussed in Section 7.2, 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 [29], 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 Section 5.3 even when handling several hundred events, and several hundred devices the latency stays less than 50 ms. We now use the Event Heap for transportation of mouse events for the Macintosh port of PointRight and it allows several users to smoothly control the mouse over a 802.11b wireless network.

Based on this experience 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 different communication libraries to program applications for an interactive workspace.

9 Related Work

A large number of interesting and complex, but non-interoperable, projects [5, 10, 13, 18, 45] are investigating room or work-area based ubiquitous computing. Each has uncovered important insights in ubiquitous computing but none have yet to propagate and deploy their frameworks significantly beyond the project's boundaries. Many are focused more on making the environment 'smart' and responsive to users' needs, and have focused less on creating a reusable, portable and robust software infrastructure.

The Gaia project [12] is seeking to make an operating system for interactive-workspace-like environments. 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.

The i-Land project [45] has been investigating human computer interaction in ubiquitous computing rooms. As a platform for creating and testing applications in their prototype room, they have created the BEACH system [46, 47]. It provides both a conceptual model of applications in such spaces, and a corresponding software infrastructure based on SmallTalk. In their system, coordination is accomplished through the use of shared objects and the ability to monitor those objects for state changes. This is most similar to a publish-subscribe system, and has the advantages and disadvantages of that system as described in Section 6.2.

The Intelligent Room project at MIT is also investigating environments similar to those we are investigating, although they are focusing on making the environment smart and responsive to the users in the space. They have created the Metaglue system [13] as a framework for creating the applications in their space. It is Java based and relies on RMI for inter-application communication. It most resembles the RMI-Rendezvous system described in Section 6.1. To lessen the coupling inherent to RMI, calls in their system are routed through an intermediary that forwards them to the appropriate final target. This allows the software infrastructure to switch the bindings between objects on the fly.

Hasha [24] proposes publish/subscribe for controlling homes filled with smart appliances, sensors and I/O devices; we believe the limited temporal persistence and expiration properties we have added to tuplespaces make them slightly more useful for connecting legacy components and applications and for dealing with partial failure and state corruption. Jini [48] provides lower level mechanisms by which clients and servers that understand common interfaces can interact with each other, but does not specify how coordination proceeds after the initial rendezvous. JavaSpaces [3], Gigaspaces [1] (a commercial implementation of JavaSpaces), and TSpaces [50] are all extended versions of the tuplespace model, and an earlier implementation of the Event Heap was actually built on top of TSpaces.

10 Conclusions

Because of practicality and complexity constraints, many ubiquitous computing application scenarios, both within and without interactive workspaces, will continue to be characterized as loosely-integrated ensembles of heterogeneous, and often legacy, components. We propose that at least within the sub-domain of interactive workspaces, a tuplespace model with extensions does the best job of satisfying the demands and characteristics of the space. Further, we suggest that this model gives a way of thinking about how applications for interactive workspaces can be written and made to interact with one another in a standard manner.

While we have yet to work outside of this sub-domain, our survey of other work makes us think that a similar set of coordination infrastructure properties may apply more generally in the ubiquitous computing field. This domain may be a "killer app" for tuplespace-based models of coordination, because of the basic model's portability, extensibility, flexibility, and ability to deal with heterogeneous environments. We encourage interested readers to help evaluate and extend our approach by setting up their own Interactive Workspace. The hardware can be easily simulated using a collection of PC's, handhelds and laptops, and the Event Heap software described in this paper, along with other software infrastructure for interactive workspaces that we have developed, are available as Open Source at **http://iros.sourceforge.net**. More information on the Interactive Workspaces project in general may be found at **http://iwork.stanford.edu**.

Acknowledgments

First, we give our thanks to the reviewers who have helped to make this paper much stronger. This work wouldn't have come to fruition without the help of the many other participants in the Interactive Workspaces project. In particular, we would like to thank Pat Hanrahan and Terry Winograd who helped us in thinking through many of the issues of coordination in interactive workspaces as the Event

Heap was being developed. Thanks also to the past and current members of the infrastructure sub-group who have always provided a good sounding board for ideas. The work described here has been supported by DOE grant B504665, grants from the Wallenberg Foundation, the Stanford Networking Research Center (SNRC), and by donations of equipment and software from Intel Corp., InFocus, IBM Corp. and Microsoft Corp.

References

1. *GigaSpaces Platform*. White Paper, . 2002, New York, NY, USA: GigaSpaces Technologies Ltd. 14. http://www.j-spaces.com/download/GigaSpacesWhitePaper.pdf (Verified 10/2002).

2. *Gryphon Project Home Page*, , IBM Research http://www.research.ibm.com/gryphon/home.html (Verified: 10/2002).

3. *JavaSpaces Service Specification*, . 2000, Palo Alto, CA: SUN Microsystems. http://java.sun.com/products/javaspaces (Verified: 10/2002).

4. *MQSeries: Message Oriented Middleware*, : IBM. http://www-3.ibm.com/software/ts/mqseries/library/whitepapers/mqover/ (Verified: 10/2002).

5. 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.

6. 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.

7. Ahuja, S., N. Carriero, and D. Gelernter, *Linda and friends.* Computer, 1986. **19**(8): p. 26-34.

8. Anderson, J.R., *Learning and memory : an integrated approach*. 2nd ed. 2000, New York: Wiley. xviii, 487.

9. 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.

10. 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.

11. 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.

12. Cerqueira, R., *et al. Gaia: A Development Infrastructure for Active Spaces*. in *Ubitools Workshop at Ubicomp 2001*. 2001. Atlanta, GA.

13. 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.

14. 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.

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

16. 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.

17. Edwards, W.K. and R. Grinter. *At Home with Ubiquitous Computing: Seven Challenges*. in *Ubicomp 2001*. 2001. Atlanta, GA, USA.

18. 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.

19. 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).

20. Fox, A., *et al.*, *Integrating information appliances into an interactive workspace.* IEEE Computer Graphics and Applications, 2000. **20**(3): p. 54-65.

21. Gelernter, D. and N. Carriero, *Coordination languages and their significance.* Communications of the ACM, 1992. **35**(2): p. 97-107.

22. 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.

23. 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.

24. Hasha, R., *Needed: A Common Distributed Object Platform*, in *IEEE Intelligent Systems*. 1999. p. 14-16

25. Humphreys, G., *et al.*, *WireGL: A scalable graphics system for clusters.* Proceedings of the ACM SIGGRAPH Conference on Computer Graphics, 2001: p. 129-140.

26. Johansen, R., *Leading business teams : how teams can use technology and group process tools to enhance performance*. 1991, Reading, Mass.: Addison-Wesley. xxiv, 216.

27. 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.

28. 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.

29. 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.

30. Johanson, B., *et al. Multibrowsing: Moving Web Content across Multiple Displays*. in *Ubicomp 2001*. 2001. Atlanta, GA, USA.

31. Kindberg, T. and A. Fox, *System Software for Ubiquitous Computing*, in *IEEE Pervasive Computing*. 2002. p. 70-81

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

33. 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.

34. Mark, G., *Collaborative Design Within and Between Warrooms.* submitted to the Human-Computer Interaction Journal, 2002.

35. Mark, G., *Extreme Collaboration.* Communications of the ACM, 2002. **45**(4).

36. Miller, R. *Response Time in Man-Computer Conversational Transactions*. in *AFIPS Fall Joint Computer Conference*. 1968.

37. Munson, M., *System Support for Composing Distributed Applications Using Events*. Diploma Dissertation, Department of Computer Science. 1998, Cambridge, UK: Cambridge University.

38. 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.

39. 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.

40. 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.

41. 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.

42. Ponnekanti, S., *et al. ICrafter: A Service Framework for Ubiquitous Computing Environments*. in *Ubicomp 2001*. 2001. Atlanta, GA, USA.

43. 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.

44. 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.

45. 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.

46. Tandler, P. *The BEACH Application Model and Software Framework for Synchronous Collaboration in Ubiquitous Computing Environments.* in *UbiTools '01 Workshop at Ubicomp 2001*. 2001. Atlanta, GA.

47. Tandler, P. *Software Infrastructure for Ubiquitous Computing Environments: Supporting Synchronous Collaboration with Heterogeneous Devices*. in *Ubicomp 2001*. 2001. Atlanta, GA, USA.

48. Waldo, J., *The Jini architecture for network-centric computing.* Communications of the ACM, 1999. **42**(7): p. 76-82.

49. Weiser, M., *The computer for the 21st century.* Scientific American, 1991. **265**(3): p. 66-75.

50. Wyckoff, P., *et al.*, *T spaces.* IBM Systems Journal, 1998. **37**(3): p. 454-74.

51. 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.