

Portability, Extensibility and Robustness in iROS

Shankar R. Ponnekanti, Brad Johanson, Emre Kıcıman and Armando Fox
Computer Science Dept
Stanford University
Stanford, CA 94305
{pshankar@cs, bjohanso@graphics, emrek@cs, fox@cs}.stanford.edu

Abstract

The dynamism and heterogeneity in ubicomp environments on both short and long time scales implies that middleware platforms for these environments need to be designed ground up for portability, extensibility and robustness. In this paper, we describe how we met these requirements in iROS, a middleware platform for a class of ubicomp environments, through the use of three guiding principles - economy of mechanism, client simplicity and levels of indirection. Apart from theoretical arguments and experimental results, experience through several deployments with a variety of apps, in most cases not done by the original designers of the system, provides some validation in practice that the design decisions have in fact resulted in the intended portability, extensibility and robustness. A retrospective examination of the system leads us to the following lesson: A logically-centralized design and physically-centralized implementation enables the best behavior in terms of extensibility and portability along with ease of administration, and sufficient behavior in terms of scalability and robustness.

1 Introduction

This paper concerns the design of middleware for ubiquitous computing (ubicomp) environments. While demarcating ubicomp from mobile and distributed computing, Kindberg and Fox [19] state the following principle:

Volatility principle (VP): The set of participating users, hardware and software components in a ubicomp environment is highly dynamic and cannot be predicted in advance. The sudden departure or arrival of a service, device, or user should be considered normal operation, not an exceptional condition or a failure requiring special handling.

The volatility principle has serious implications for ubicomp middleware platforms. On larger time scales, volatility implies that incremental evolution/accretion and therefore extreme heterogeneity (in terms of the hardware/OS technologies as well as the environment configurations) will be the norm in these environments [8]. On shorter time scales, it implies that partial failures will be the "common case". Thus, to effectively address VP, ubicomp middleware frameworks must meet the following requirements:

- Platform portability including legacy support (R1): OS and hardware heterogeneity implies that the middleware platform itself must be portable across different hardware and OS technologies. "Java everywhere" and similar approaches do not suffice, because they attempt to define heterogeneity out of existence and assume that non-conforming applications will be rewritten. Furthermore, due to the existence of useful legacy software such as the Web, desktop/productivity applications, etc., ubicomp software must make it easy to integrate legacy applications.
- Application portability and new device extensibility (R2): Ubicomp environments are characterized by extreme diversity, and no two ubicomp environments are likely to be identical with respect to the available resources and their configurations. Applications written atop the platform should be easy to port and adapt to different environments. To accommodate incremental evolution, extending applications by adding new devices should be easy.
- Robustness and ease of administration (R3). Volatility on smaller time scales requires us to deal with dynamism (e.g. people or devices entering/leaving spaces without signoff) and partial failures as common cases. Transient failures in parts of the system should not cause cascading failures, and recovery from transient failures should not require unavailability or recovery of the whole system. Further, the lack of well-qualified

system administrators in ubicomp environments implies that the middleware software should be easy to administer.

In this paper, we examine how we met these requirements in **iROS**, a middleware system for *interactive workspaces*, a particular class of ubicomp environments. In accordance with the needs of interactive workspaces, iROS consists of three subsystems: EventHeap for application coordination, DataHeap for data movement and transformation, and ICrafter for user control of resources. Though we have described two out of the above three subsystems individually (ICrafter [23], EventHeap [15]), to date we have not described in detail nor quantitatively evaluated the systemwide portability, extensibility, and robustness, which result from the *synergistic combination* of the three subsystems.

iROS was previously introduced in [16], an overview article describing the middleware and HCI issues in interactive workspaces, that only briefly and informally discusses the middleware components. Here, we describe and analyze in detail the key design principles that enabled iROS to achieve the above requirements, and demonstrate using new quantitative evaluation results that these principles are effective in practice. We also derive the lesson that for room-sized ubicomp systems, centralized infrastructure-based mechanisms enable several systemwide behaviors that are necessary/desirable, while providing sufficient scalability.

The rest of the paper is organized as follows. In section 2, we briefly introduce our specific problem domain, interactive workspaces, and the various subsystems of iROS. In sections 3-5, we explain the design principles employed by iROS to address R1, R2, and R3 respectively. Section 6 derives suitable lessons based on the results of sections 3-5. In sections 7 and 8, we survey related work and conclude.

2 Interactive Workspaces and iROS

As an example of a ubicomp environment, we focus on an *interactive workspace (IW)*: a localized technology-augmented environment where people come together for collaborative work. Our testbed, the iRoom (figure 1), features three rear projected touch-sensitive screens along one wall, a bottom projected table, and a custom 12-projector tiled display (“the Mural” [14]) driven by a workstation cluster that does distributed rendering of OpenGL.

iROS is the software infrastructure for interactive workspaces designed based on the requirements of these environments. The programming model for iROS is one of ensembles of independent entities that communicate via message passing (“events”) using a logically-centralized, broadcast-based communication substrate



Figure 1. A meeting in the the iRoom

called the EventHeap [15]. The EventHeap is based on the tuplespaces model first proposed by LINDA [10], although unlike LINDA, events in the EventHeap have timeouts (similar to TSpaces [29]) to prevent unlimited accumulation of events. The EventHeap also makes a number of other modifications to the basic tuplespaces framework, as described in [15].

The DataHeap, the second iROS component, provides type-independent and location-independent storage of large and semi-permanent data in an interactive workspace. To store data in the DataHeap, applications submit the data (such as a document) and associated metadata (owner, creation time, etc) to the DataHeap. To retrieve data, applications can query the DataHeap based on the metadata. Where necessary, the DataHeap also provides type transformation, using a type transformation system called Paths [18].

The final component of iROS is ICrafter [23]: a framework for *services*. We refer to any hardware or software entity (lights, projectors, media players, a browser/PowerPoint running on a large display, etc.) that is controllable over the network as a service. Services written using the ICrafter framework can be programmatically controlled by applications or directly controlled by end-users (from a web browser, for example).

In the following sections, we examine how the requirements R1, R2 and R3 were met in iROS. In general, we focus on the design principles, but use iROS to illustrate and evaluate the effectiveness of these principles in practice.

3 R1: Platform Portability

A simple strategy to simplify portability is to reduce the number of mechanisms that need to be ported to every client platform - a principle we call *economy of mechanism*. Consider application coordination: table 1 shows the three types of application coordination that have been noticed by us

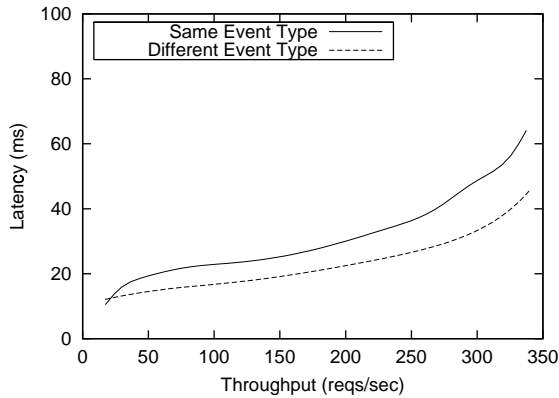


Figure 2. Latency vs Throughput plot of the EventHeap. 100 clients were used to generate a variable “background” request rate, while a separate client was used for the latency probe. In the solid curve, latency probes and background events were of the same event type, whereas different event types were used for the dashed curve.

and other ubicomp researchers. Many ubicomp frameworks provide multiple mechanisms for supporting the different types of coordination. For example, Jini provides RMI, JavaSpaces and an event notification mechanism. In contrast, we use a single mechanism - the EventHeap - for all the above types of coordination.¹

A single mechanism implies less to port across all the client platforms. Consequently, it was easy to port the EventHeap client library to multiple platforms (UNIX, Windows, Mac, WinCE²) and languages (Java, C/C++, Python). As shall be explained in the next section, economy of mechanism has also offered another advantage – dynamic extensibility through interposition.

The use of a single mechanism raises scalability concerns. To test this aspect of the system, we evaluated the scalability of the EventHeap, the results of which are shown in figure 2. The figure illustrates that we achieve sufficient scalability for room-sized ubicomp environments, indicating that scalability is not a concern *for our domain*.

Apart from simplicity at the logical level that results from a single, simple coordination mechanism, iROS reduces client complexity at the implementation level too. The EventHeap, DataHeap, and ICrafter were each implemented for *client simplicity*:

- The EventHeap implementation is client-server based, with the event buffering and matching logic handled

¹Applications such as streaming that must use a point-to-point connection can set up such a connection after initial coordination over the EventHeap.

²Currently underway

by the server.

- The DataHeap stores both the data and metadata on the server-side – the data on a WebDAV [1] server and the corresponding metadata (including the datatype) in a fast in-memory XML database. All the data transformation functionality is concentrated on the server-side too.
- ICrafter places UI selection, generation, and adaptation functionality in an infrastructure-based service called the interface manager (IM). Client devices (end-user devices) simply request UI’s from the IM while specifying the target service and the desired toolkit (HTML/WML/VoiceXML browser, Java Swing etc). The IM selects a suitable UI generator for the target service and UI toolkit from its repository of UI generators. In fact for some toolkits (such as HTML, Swing), the IM automatically generates a functional (if clumsy) if a handwritten UI is not found in the repository for the given service.

Placing much of the complexity on the server-side implies ease of porting to various client platforms, especially to resource-constrained clients, which are expected to be a major component of ubicomp environments. Placing functionality on the server-side also has the obvious downside of requiring a server, which may not often be feasible for ad hoc environments. However, for fixed environments such as interactive workspaces, such a server is readily available.

Another advantage resulting from economy of mechanism and client simplicity is legacy support. In general, the more a framework expects from the underlying platform and participating applications, the harder it is to integrate legacy platforms and applications, because legacy systems provide limited maneuverability. The simplicity of iROS client mechanisms implies that the “bar is set very low” for integration, and this has contributed to the ease of legacy platform and application integration. For example, our collaborators in the civil engineering department were able to integrate their legacy construction data viewers into the EventHeap easily [15]. Modifying the original standalone viewers to use the EventHeap required no more than about 100 lines of code each. As another example, using a Java-COM bridge, we wrapped Microsoft IE into an ICrafter service. Creating a simple version of the service (that only supports the “gotoURL” method) requires just 20 semicolons of Java code. This service allows users to send Web pages to displays by sending a suitable navigate command to the ICrafter IE service running on that display. We call this behavior multibrowsing [17].

Table 1. Types of coordination behavior

Coordination type	Explanation	Possible modes	Example
Anonymous, event-driven coordination	Sender sends events to notify changes of state and other significant occurrences. Sender often unaware of who/how many receivers subscribe.	one-one, one-many, one-none	A motion sensor sends an event when it detects motion in the room. Interested applications react accordingly.
Intentional naming	Sender identifies receiver(s) through attributes in the message	one-one, one-many	An application requests a document to be displayed on all the “large” displays in the room.
Point-to-point	Sender explicitly addresses the message to receiver	one-one	An application requests a browser running on a specific display to navigate to a particular URL.

4 R2: Application Extensibility

An important aspect of our design is *levels of indirection (LoI)* at multiple levels of the architecture - in communication, data exchange, and user control.

The DataHeap provides an LoI between data senders and receivers. Data producers store documents and associated metadata in the DataHeap, and consumers query based on metadata and indicate which formats they can accept. If the format indicated by a receiver does not match the original data type, the DataHeap dynamically instantiates a chain of transformation operators to convert the data to one of the acceptable types. Hence, the Data Heap frees data producers from having to know who the consumers of their data will be. This property is essential for extensibility to new devices, and avoids having to rewrite existing applications to support the data formats required by new devices.

ICrafter provides an LoI between services and their controllers in the form of the IM, and this LoI facilitates extensibility to new devices possessing new UI toolkits (WML, VoiceXML, etc). To allow a service to be controlled by a new toolkit, a UI generator for the service for the appropriate toolkit can be added at the IM, and no modification to the service is necessary. As explained in [24], the IM can also be configured to automatically search a web-based global repository for new toolkit service UI’s. Thus, when a new UI toolkit (such as WML or VoiceXML) appears, the IM automatically searches for the new UI-toolkit generators for all (and only) the services installed in the environment. This has the effect that services automatically adapt to control devices with new UI toolkits.

To illustrate how the levels of indirection in DataHeap and ICrafter contribute to the extensibility of applications, we describe a sample application called SmartPresenter – a multi-display, multi-object presentation program for interactive workspaces.

While traditional presentation programs coordinate the display of slides across time, SmartPresenter coordinates the display of information across both time and display surfaces. For instance, in the iRoom, with three large displays on one wall, for some specific point in their presentation,

the presenter may configure the system to display an outline of the talk on the left-most display, the main content slide on the middle display, and a detail of a dataset on the right-most display.

The core of the SmartPresenter application is the SmartPresenter service that reads a presentation script specifying what actions should be taken at what point in the presentation. The most common action is displaying a particular data object on a named display such as slide #4 of a PowerPoint presentation, or a digital photograph. (Note that every display runs a display service instance, and each instance has a unique name.) The SmartPresenter service reads the script and issues appropriate commands (over the EventHeap) to the individual display services.

One of our new displays, the Mural, cannot display Microsoft PowerPoint presentations but can display JPEG images. However, by adding a simple PowerPoint-to-JPEG transformer (written using PowerPoint’s ActiveX API) to the DataHeap, the Mural could be integrated into SmartPresenter without changing the SmartPresenter or the Mural. Consequently, when a user asks to display a presentation on the Mural, the alternative JPEG version is shown.

By default, SmartPresenter only provides web-based (HTML) control using ICrafter’s automatic HTML UI generator. However, new modes of presentation control, such as through Java Swing, WML, and VoiceXML can be easily enabled by ICrafter. These tasks involve only adding the corresponding UI generators for SmartPresenter at the IM, without the need for modifying any of the existing SmartPresenter code, and without installing any SmartPresenter specific code on the new devices. For example, we wrote a SUIML³ generator for SmartPresenter using only 75 lines of XML.

The economy of mechanism principle described in the previous section also contributes to extensibility. The fact that all applications use a single broadcast mechanism for all their coordination needs implies the possibility of *snooping and intermediation*. That is, since events are always sent

³SUIML (Swing UI Markup Language) is a homegrown XML-based language for describing Java Swing UI’s.

between applications through the EventHeap, an intermediary can observe an event from a source and generate one or more events of different types in order to cause a desired action in a different receiver or receivers. Using snooping, SmartPresenter can be extended to allow audience members to track the current presentation (on any of the displays) from a laptop. Tracking the current presentation is done by snooping on the main control command events being sent to that display. To enable this behavior, all the user needs to do is to run a display service instance on her laptop with the same name as the display service running on the desired target display. Table 2 summarizes the effort needed for various SmartPresenter extensions.

Most frameworks provide environment portability by assuming that the applications discover the services in the local environment and adapt their behavior accordingly. Interposition provides an additional degree of environment portability. To illustrate this, consider multibrowsing (recall from previous subsection) that allows applications to send web pages to target displays. Early prototype application developers had hard coded the names of target displays in the iRoom, making their applications non-portable to other iROS installations. We exploited the ability to interpose in *mbforward*, a simple intermediary that picks up multibrowse events directed to the specified targets and automatically re-routes them to different machines by generating new events. Using this mechanism, we were able to use multibrowsing demos originally hardcoded to the iRoom for demonstrations in other locations, without changing any of the original application source code.

5 R3: Robustness and Ease of Administration

Failure resilience in iROS is achieved through multiple mechanisms - LoI in communication, ICrafter’s soft state mechanisms, and EventHeap’s fast restart.

The EventHeap provides an LoI in communication via loose coupling of the communicating entities, which results in improved failure resilience. First, entities communicating through the EventHeap do not have direct connections between them (referred to as *spatial decoupling* by LINDA proponents) encouraging failure resilience through isolation. Second, since events are semi-persistent, communicating entities do not have to be up at the same time (referred to as *temporal decoupling* by LINDA proponents). Temporal decoupling can mask transient failures in entities. In particular, if a service to which an event is directed temporarily dies and is restarted immediately (by an enclosing “while{1} restart” script), the service still picks up the event, and the sender does not perceive a failure.

Services in ICrafter advertise their presence and other state information with periodic beacon events. The expira-

tion time of a beacon event is set to twice the beacon period. The beaconing library used by services can also be used by components other than services, such as short-running applications to announce their state information. The beaconing library provides a soft state mechanism for state sharing among application components and services. “Stale” beacon events associated with failed components will eventually expire and other components will detect their absence after at most two beacon periods.

The most involved partial failure scenario is the crash of the centralized EventHeap server – potentially a single point of failure that can in turn cause cascading failures as other components lose their connections to the server. We prevent this “single point of failure” behavior by a synergistic combination of fast restart, auto-reconnect and beaconing as described below:

- To simplify recovery and enable faster performance in the steady state, the server does not write events to disk. As a result, we may lose some events during a crash, but the EventHeap can be restarted quickly without any special recovery actions, and the restart time for the server itself is only 200 milliseconds.⁴ The lost events can cause temporary disruption (e.g., a light control command will have no effect) but retrying the command after the EventHeap has recovered fixes the problem.
- Further, the EventHeap client library provides an auto-reconnect feature: connected applications detect an EventHeap failure and they auto-reconnect when it is restarted.
- Some inconsistency is expected for a brief period following the restart of the EventHeap because all the built up soft state is lost in the crash. However, this state is automatically replenished in at most one beacon period after the clients reconnect.

Thus, the total time for recovery as perceived by the user is $TTR = T_{JVM} + T_{EH} + T_{RC} + T_B$, where T_{JVM} is the time to start the JVM, T_{EH} is the time for EventHeap initialization, T_{RC} is the time for all the clients to reconnect, and T_B is a beacon period. Typically T_{JVM} is between 1.5 and 2.5 seconds and $T_{EH} = 200ms$.

Consistent with Miller [20], we define “fast enough” recovery as 10 seconds, which according to Miller’s study is noticeable but unlikely to distract the user from the task at hand. Figure 3 shows the reconnect times for clients (T_{RC}) under varying values of the number of active clients N at the time the EventHeap fails. From the figure, it may be

⁴Placing the Event Heap startup command inside a `while(1)` loop recovers from JVM crashes; we are working on external monitoring to restart the Event Heap when livelock or thrashing is detected.

Table 2. Effort needed to extend SmartPresenter in various ways

SmartPresenter Task	iROS features used	Number of semicolons
Adding Mural to SmartPresenter	DataHeap transformer API, Third-party Java-COM bridge	84 semicolons + 46 lines XML
Web-based control of SmartPresenter	ICrafter automatic HTML UI generator	Free
Swing-based control of SmartPresenter	Add SUIML UI generator for SmartPresenter to IM's repository	75 lines of XML
Allowing client laptop to follow presentation	EventHeap snooping	Free

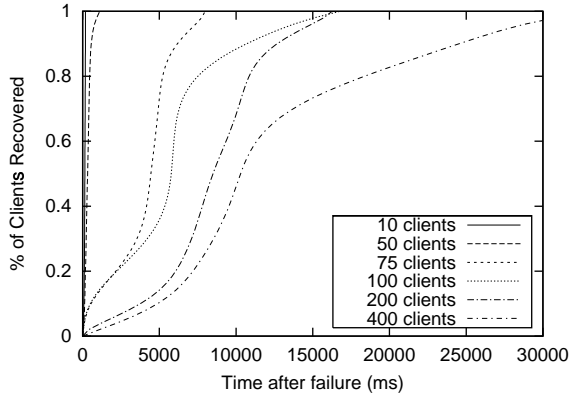


Figure 3. Speed of EventHeap recovery with different numbers of clients. The figure plots the fraction of clients successfully reconnected as a function of time.

inferred that for our typical operating parameters (less than 50 simultaneous clients and a beacon period of 5 seconds), $TTR < 2.5 + 0.2 + 1.2 + 5$, i.e., less than 9 seconds. This recovery time is currently adequate for our purposes, but we are exploring techniques for improving reconnect time when more than 100 clients are connected.

The auto-reconnect feature plays a key role in enabling dependency-free restarts of failed components. Without this, we would need to restart *all* iROS components, as well as all iRoom services and applications, after an EventHeap crash. In fact, we had this problem with an earlier version of the EventHeap based on IBM TSpaces [29].

Below, we summarize the failure resilience features of iROS:

- If a service experiences a transient failure and is immediately restarted, it can still pick up events directed to it (assuming the event has not expired yet), and thus the transient failure is masked.
- If a service (or an application component) fails permanently, its beacon events eventually expire causing other entities to detect its failure in at most two beacon

periods.

- If the EventHeap itself has a transient failure, it can be restarted quickly and beacons restore the soft state within a beacon period.

We do not argue that these are the *only* recovery mechanisms needed in an interactive workspace—these do not handle deterministic failures, such as a pathological event that always crashes the EventHeap, or hard failures, such as a persistent hardware failure on one of the machines. But these mechanisms do handle a wide variety of transient failures, and we have verified from experience that most observed failures of iRoom software are in fact transient and curable through restarts.

With respect to administration, it may appear that our strategy of using centralized server-based mechanisms for client simplicity implies additional administration. However, since recovery of the EventHeap doesn't require any special actions, it can often be automated, and when manual intervention is necessary, it can be performed by "anyone", and a qualified administrator is not necessary. More importantly, client simplicity actually results in significant software administration/maintainability benefits because:

- Less functionality on clients implies less to install on the numerous clients and fewer upgrades.
- Policy configurations are centralized on the server, and hence are easier to maintain.

Historical experience of corporate enterprises indicates that centralized administration and simpler client software leads to simpler administration and reduced total cost of ownership.

6 Synthesis

Table 3 summarizes the various principles employed by iROS's subsystems to deal with portability, extensibility and robustness. Modifying any of these design choices would affect multiple requirements. Note that all the design choices are related to centralization - either at the logical

Principle	Architectural feature	Benefits	Reference
Economy of mechanism	One mechanism for app coordination Snooping and interposition	Less to port and ease of integrating legacy systems (R1)	Section 3
		Environment portability (R2)	Section 4
Client simplicity	Complex functionality on server in each of DataHeap, EventHeap and ICrafter	Less to port on each client device (R1)	Section 3
		Ease of software administration and maintenance (R3)	Section 5
Levels of indirection or LoI	Spatial and temporal decoupling in EventHeap Interface Manager (LoI for UI's) DataHeap (LoI for data exchange)	Failure resilience (R3)	Section 5
		Extensibility to new devices (R2)	Section 4
		Extensibility to new devices (R2)	Section 4

Table 3. Design choices/principles and how they address the requirements. Each principle affects multiple requirements. Note that all the design choices in the left column are related to either logical centralization or a centralized implementation of one or more iROS subsystems.

level or the implementation level. In other words, centralization played a key role in achieving the requirements we set out with. Centralization is not panacea – table 4 summarizes some negative implications that stem from centralization – but these disadvantages are either not relevant for our domain or can be effectively neutralized as shown in the table. Thus, we observe that centralization provides a simple way of achieving many of the properties that are necessary in this domain.

Apart from the theoretical arguments and experimental results presented in this paper, our positive deployment experiences with iROS confirm the validity of the design principles. iROS is a real system in daily use by multiple groups of non-systems researchers. Regular group meetings in the iRoom routinely use several iROS applications and services (many of which are described in the overview article [16]). iROS has also been deployed in more than half a dozen environments, several of which are non-CS environments such as the Center for Integrated Facilities Engineering (<http://cife.stanford.edu>) and the Program in Writing and Rhetoric. iROS is expected to be the base technology for new distance-learning classrooms to be completed in 2003. Although the deployments have been far from perfect (we describe areas of future work below), iROS has been sturdy under a variety of conditions of use by people other than its creators. We believe that the positive deployment experiences with iROS validate the choice of abstractions and requirements.

Two important avenues of future work are a comprehensive security model for iROS and better detection of failures. A drawback of the event-driven anonymous coordination we exploit for application coordination is that it is not meaningful to talk about end-to-end delivery semantics of messages, since the sender does not know in advance who the receiver(s) will be or whether there will be any at all.

As a result, failures in receiver(s) are harder to track down. With respect to security, we note that a centralized architecture leads to a simpler security solution since it implies a single place for access control and policy management.

7 Related Work

Table 5 compares iROS to other most closely related ubiquitous computing architectures in their support for platform and language portability, application portability and extensibility, and resilience to partial failures. We omit detailed descriptions of these systems due to lack of space. Jini [3] and UPnP [27] are network-level frameworks for service discovery and interoperation; and Gaia [25], One.World [12], and Equip [11] are higher-level ubiquitous computing architectures or meta-operating systems. We base our comparison on the requirements R1-R3, and attempt to compare how the mechanics of these systems meet these requirements. In particular, we are not interested in comparing the choice of functionalities offered by each of these systems but are instead comparing whether the design and implementation of those functionalities (and the applications built to use them) provides for portability across environments, extendibility to new devices, and resilience to partial failures. It is worth noting that some of these systems provide benefits that iROS does not, such as One.World's support for migration, and Gaia's context service.

The Intelligent Room at the MIT AI Laboratory [6] and Microsoft Research EasyLiving [4] both use a combination of sophisticated sensor fusion and AI techniques to enable the environment to deduce the user's needs from contextual and other cues. "Smartness" was not one of our goals: we focused instead on providing the infrastructure for application programmers to simplify writing applications with the

Negative Implication	Offsetting factor	Section
Scalability	Centralized systems can achieve sufficient scalability for this domain.	See graph 2
Single point of failure	Fast restart	See graph 3
Requires a server	Not a concern for most home/office environments and interactive workspaces in particular. Potential concern in ad hoc environments.	Section 3

Table 4. Negative implications of centralization and corresponding offsetting factors. For the interactive workspaces domain, the negative implications can be effectively offset.

	R1: Platform and language portability	R2: Application portability and extensibility	R3: Recovery from partial failures
Jini	No	No	Reclaims resources
UPnP	Yes	No	No
Gaia	Yes	Environment portability	Reclaims resources
One.World	No	Partial	Yes
Equip	No	Partial	?

Table 5. Summary evaluation of the support other ubiquitous computing architectures provide for platform and language portability, application portability and extensibility, and resilience to partial failures. Next to iROS, Gaia provides the best overall support; One.World provides the most support for recovery; and UPnP provides good platform and language portability.

behaviors they desire.

The Beach architecture [26] is built for synchronous collaboration among users of “roomware”, such as tables and chairs integrated with information technology. Beach, implemented in SmallTalk, provides a sophisticated layered software architecture for developing applications in this environment, using object-oriented language techniques to provide extensibility of applications and reusability of components.

The EventHeap draws upon pioneering earlier work by the proponents of the tuplespaces (LINDA [10]) and publish-subscribe (InfoBus [22]) frameworks. ICrafter improves upon earlier work by Hodes et al [13], while the DataHeap builds upon prior work in datatype transformation (TOM [21]) and attribute-based filesystems (Presto [7]).

Our approach to robustness based on fast restart draws from the recursive restartability project [5]. The use of beaconing-based soft state is a well known technique in the systems community. Recent projects in related domains such as INS [2] and SNS/TACC [9] have also exploited this technique for increased robustness. Finally, in [28], Wang et al. describe their experience improving the dependability of home networking technologies using redundant communication networks (power-lines, phone-lines and RF). Network-level reliability was not one of our goals however: we focused on robustness of our system to failures of the components themselves, and assume the existence of reasonable best-effort network technologies.

8 Conclusions

In this paper, we studied how portability, extensibility and robustness were achieved in iROS, a middleware platform for a class of ubicomp environments. The three key design principles underlying iROS that have facilitated portability, extensibility and robustness are listed below:

1. Economy of mechanism: A single mechanism for all types of application coordination.
2. Client simplicity: Putting complexity on infrastructure based servers.
3. Levels of indirection: A level of indirection – whether in data exchange, user control or communication – allows us to add new behaviors at the indirection point without changing the end-points.

We observe that centralized design and implementation facilitate applying each of these design principles. We have also shown how the disadvantages of centralization are either not relevant or can be effectively offset in this domain. In particular, centralized systems can achieve sufficient scalability for this domain; and the combination of soft state and fast restart neutralize the single point of failure. Thus, we conclude that with a logically-centralized design and physically-centralized implementation, we get the best behavior in terms of extensibility and portability along with ease of administration, and *sufficient* behavior in terms of scalability and robustness; any change in that set of de-

sign decisions would have a negative effect on more than one of the desired properties.

References

- [1] Web-based Distributed Authoring and Versioning. Available at <http://www.webdav.org>.
- [2] W. Adjie-Winoto, E. Schwartz, H. Balakrishnan, and J. Lilley. The Design and Implementation of an Intentional Naming System. In *Proceedings of the 17th ACM Symposium on Operating Systems Principles (SOSP-17)*, volume 33, pages 186–201, December 1999.
- [3] K. Arnold, B. O’Sullivan, R. W. Scheifler, J. Waldo, and A. Wollrath. *The Jini Specification*. Addison Wesley, 1999.
- [4] B. Brumitt, B. Meyers, J. Krumm, A. Kern, and S. Shafer. Easyliving: Technologies for intelligent environments. In *Handheld and Ubiquitous Computing (HUC 2000), First International Symposium*, sep 2000.
- [5] G. Candea and A. Fox. Recursive restartability: Turning the reboot sledgehammer into a scalpel. In *Eighth Workshop on Hot Topics In Operating Systems (HotOS-VIII)*, pages 110–115, Elmau, Germany, May 2001.
- [6] M. Coen. The future of human–computer interaction or how i learned to stop worrying and love my intelligent room, 1999.
- [7] P. Dourish, W. K. Edwards, A. LaMarca, and M. Salisbury. Uniform document interactions using document properties. In *ACM Computer Supported Cooperative Work*, pages 55–64, November 1999.
- [8] W. K. Edwards and R. E. Grinter. At home with ubiquitous computing: Seven challenges. In *Third International Conference on Ubiquitous Computing (UbiComp2001)*, 2001.
- [9] A. Fox, S. D. Gribble, Y. Chawathe, E. A. Brewer, and P. Gauthier. Cluster-Based Scalable Network Services. In *Proceedings of the 16th ACM Symposium on Operating Systems Principles (SOSP-16)*, St.-Malo, France, October 1997.
- [10] D. Gelernter. Generative communication in LINDA. *ACM Transactions on Programming Languages and Systems*, pages 80–112, January 1985.
- [11] C. Greenbagh. Equip: a software platform for distributed interactive systems. Technical Report Equator-02-002, Equator, April 2002.
- [12] R. Grimm et al. Systems directions for pervasive computing. In *Eighth Workshop on Hot Topics In Operating Systems (HotOS-VIII)*, pages 147–151, sep 2001.
- [13] T. D. Hodes, R. H. Katz, E. Servan-Schreiber, and L. Rowe. Composable Ad-hoc Mobile Services for Universal Interaction. In *Third ACM Conference on Mobile Computing and Networking (MobiCom 97)*, Budapest, Hungary, September 1997.
- [14] G. Humphreys, I. Buck, M. Eldridge, and P. Hanrahan. Distributed Rendering for Scalable Displays. In *IEEE Supercomputing 2000*, 2000.
- [15] B. Johanson and A. Fox. The Event Heap: A Coordination Infrastructure For Interactive Workspaces. In *Fourth IEEE Workshop on Mobile Computing Systems and Applications (WMCSA 02)*, Callicoon, NY, June 2002.
- [16] B. Johanson, A. Fox, and T. Winograd. The Interactive Workspaces Project: Experiences with Ubiquitous Computing Rooms. *IEEE Pervasive Computing Magazine*, April–June 2002.
- [17] B. Johanson, S. Ponnekanti, C. Sengupta, and A. Fox. Multi-browsing: Moving Web Content Across Multiple Displays. In *Third International Conference on Ubiquitous Computing (UbiComp2001)*, 2001.
- [18] E. Kiciman and A. Fox. Using Dynamic Mediation to Integrate COTS Entities in a Ubiquitous Computing Environment. In *Handheld and Ubiquitous Computing (HUC 2000), First International Symposium*, Sept. 2000.
- [19] T. Kindberg and A. Fox. System Software For Ubiquitous Computing. *IEEE Pervasive Computing Magazine*, 1(1):70–81, January 2002.
- [20] R. Miller. Response time in man-computer conversational transactions. In *Proc. AFIPS Fall Joint Computer Conference*, volume 33, pages 267–277, 1968.
- [21] J. Ockerbloom. *Mediating Among Diverse Data Formats*. PhD thesis, Carnegie Mellon University, January 1999.
- [22] B. Oki, M. Pfluegl, A. Siegel, and D. Skeen. The Information Bus: An Architecture for Extensible Distributed Systems. In *Proceedings of the 14th ACM Symposium on Operating Systems Principles (SOSP-14)*, pages 58–68, 1993.
- [23] S. R. Ponnekanti et al. ICrafter: A Service Framework for Ubiquitous Computing Environments. In *Third International Conference on Ubiquitous Computing (UbiComp2001)*, 2001.
- [24] S. R. Ponnekanti, L. A. Robles, and A. Fox. User Interfaces for Network Services: What, from Where, and How. In *Fourth IEEE Workshop on Mobile Computing Systems and Applications (WMCSA 02)*, Callicoon, NY, June 2002.
- [25] M. Roman, C. K. Hess, R. Cerqueira, A. Ranganathan, R. H. Campbell, and K. Nahrstedt. GaiaOS: A Middleware Infrastructure to Enable Active Spaces. *IEEE Pervasive Computing Magazine*, 2002.
- [26] P. Tandler. Architecture of beach: The software infrastructure for roomware environments. In *CSCW 2000: Workshop on Shared Environments to Support Face-to-Face Collaboration*, Philadelphia, PA, December 2002.
- [27] UPnP Forum. Universal plug and play. Available at <http://www.upnp.org>.
- [28] Y.-M. Wang, W. Russell, A. Arora, J. Xu, and R. K. Jagannathan. Towards dependable home networking: An experience report. In *Proc. Intl. Conference on Dependable Systems and Networks*, New York, New York, June 2000.
- [29] P. Wyckoff, S. McLaughry, T. Lehman, and D. Ford. TSpaces. *IBM Systems Journal*, 37(3), August 1998. Available at <http://www.almaden.ibm.com/cs/TSpaces>.