UNIVERSITY OF CALIFORNIA

Los Angeles

Panoply: Active Middleware for Managing Ubiquitous Computing Interactions

A dissertation submitted in partial satisfaction

of the requirements for the degree

Doctor of Philosophy in Computer Science

by

Kevin Francis Eustice

2008

The dissertation of Kevin Francis Eustice is approved.

[illegible]

Deborah Estrin

[illegible]

Mark Hansen

[illegible]

Richard Muntz

[illegible]

Leonard Kleinrock, Committee Co-chair

[illegible]

Peter Reiher, Committee Co-chair

University of California, Los Angeles

2008

To my family, for their love and support,

and to Alison, my muse and best friend.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGMENTS

This dissertation would not have been possible without the expert guidance of my advisors, Peter Reiher and Leonard Kleinrock. Additionally, I am grateful for the support of the other members of the Laboratory for Advanced Systems Research. In particular, many thanks are owed to V. Ramakrishna, Nam Nguyen, and Matthew Schnaider for their work on Panoply and many late night discussions. Thanks as well to Matt Beaumont, and Erik Kline for many useful suggestions as well as hours of distraction.

A number of other people have contributed to Panoply and its application suite. Thanks are due to Everett Anderson for his work with the QED simulation, to Jason Karuza for his voucher implementation, to Glenn Glazer for his work on location-limited sideband enrollment, and to Shane Markstrum and Martin Lukac for their work on QED. Generous financial support was provided by NSF grant CNS-0427748. I would also like to thank Janice Wheeler and Peter Reiher for their expert review and editing of this dissertation.

Finally, thanks are owed to my wife and occasional co-author, Alison Walker, for her constant support and love. Many of the application ideas explored in Panoply owe a great deal to our conversations.

VITA

1976	Born Turlock, California
1998-1999	Research Intern IBM Almaden Research Center San Jose, California
1999	B.S., Computer Science Honors in Computer Science Harvey Mudd College Claremont, California
2002	M.S., Computer Science University of California Los Angeles, California
1999-2003	Graduate Intern The Aerospace Corporation El Segundo, California
2004	Graduate Technical Intern Intel Corporation Hillsboro, Oregon
2007-2008	Manager, Performance and Quality NetSeer, Inc. Los Angeles, California
1999-2008	Graduate Student Researcher Laboratory for Advanced Systems Research (LASR) University of California Los Angeles, California
2008-	Software Engineer Kiha Software Seattle, Washington

PUBLICATIONS

Peter Reiher, Kevin Eustice, V. Ramakrishna, “Security and Privacy for Pervasive Computing Environments.” Security and Privacy in Mobile and Wireless Networking. Troubadour Publishing, to be published in 2008.

Kevin Eustice, V. Ramakrishna, Nam Nguyen, and Peter Reiher, “The Smart Party: A Personalized Location-aware Multimedia Experience,” Proceedings of the Fifth IEEE Consumer Communications and Networking Conference (CCNC 2008).

Matthew Beaumont-Gaye, Kevin Eustice, and Peter Reiher, “Information Protection via Environmental Data Tethers,” Proceedings of the New Security Paradigms Workshop (NSPW) 2007.

Kevin Eustice, V. Ramakrishna, Matthew Schnaider, Nam Nguyen, Alison Walker, Peter Reiher, “nan0Spheres: Location-Driven Fiction for Nomadic User Groups,” Proceedings of the 2007 International Conference on Human Computer Interaction (HCII'07).

V. Ramakrishna, Kevin Eustice, and Matthew Schnaider, “Approaches for Ensuring Security and Privacy in Unplanned Ubiquitous Computing Interactions.” Mobile and Wireless Network Security and Privacy. Ed. Makki et al. 2007.

V. Ramakrishna, Kevin Eustice, and Peter Reiher, “Negotiating Agreements Using Policies in Ubiquitous Computing Scenarios,” Proceedings of the 2007 IEEE International Conference on Service-Oriented Computing and Applications (SOCA 2007).

V. Ramakrishna, Max Robinson, Kevin Eustice and Peter Reiher, “An Active Self-Optimizing Multiplayer Gaming Architecture,” Cluster Computing Journal, Vol. 9, No. 2, 2006.

V. Ramakrishna, Kevin Eustice, and Matthew Schnaider, “Approaches for Ensuring Security and Privacy in Unplanned Ubiquitous Computing Interactions,” Proceedings of the 2006 Intl. Workshop on Security and Privacy for Wireless and Mobile Networks.

Everett Anderson, Kevin Eustice, Shane Markstrum, Mark Hansen, and Peter Reiher, "Mobile Contagion: Simulation of Infection and Disease," Proceedings of Symposium on Measurement, Modeling, and Simulation of Malware, June 2005.

Kevin Eustice, Leonard Kleinrock, Shane Markstrum, Gerald Popek, V. Ramakrishna and Peter Reiher, “WiFi Nomads and their insecure devices: The Case for Quarantine, Examination, and Decontamination,” Proceedings of the New Security Paradigms Workshop (NSPW) 2003. Selected as one of four “Highlight Papers of NSPW 2003” and presented at ACSAC 2003.

Kevin Eustice, Leonard Kleinrock, Shane Markstrum, Gerald Popek, V. Ramakrishna and Peter Reiher, “Enabling Secure Ubiquitous Interactions,” Proceedings of the International Workshop on Middleware for Pervasive and Ad hoc Computing (MPAC) held in conjunction with Middleware 2003.

V. Ramakrishna, Max Robinson, Kevin Eustice and Peter Reiher, “An Active Self-Optimizing Multiplayer Gaming Architecture,” Proceedings of the Adaptive Middleware Services Workshop (AMSW) 2003.

Vincent Ferreria, Alexey Rudenko, Kevin Eustice, Richard Guy, V. Ramakrishna and Peter Reiher, "Panda: Middleware to Provide the Benefits of Active Networks to Legacy Applications," Proceedings of *DANCE 02*, May 2002.

Mark Yarvis, Peter Reiher, Kevin Eustice, and Gerald J. Popek. Conductor: Enabling distributed adaptation. UCLA Tech Report CSD-TR-010025, June 2001.

Peter Reiher, Kevin Eustice and Kai-Min Sung, "Adapting Encrypted Data Streams in Open Architectures," Proceedings of the *Adaptive Middleware Services Workshop (AMSW) 2001*, August 2001.

Kevin Eustice, Toby Lehman, Armando Morales, Michelle Munson, Stefan Edlund and Miguel Guillen. “A Universal Information Appliance,” IBM Systems Journal, 1999.

ABSTRACT OF THE DISSERTATION

Panoply: Active Middleware for Managing

Ubiquitous Computing Interactions

by

Kevin Francis Eustice

Doctor of Philosophy in Computer Science

University of California, Los Angeles, 2008

Professor Peter Reiher, Co-chair

Professor Leonard Kleinrock, Co-chair

In the near future, our mobile devices will act as our personal representatives in the digital domain. As users move from place to place, their devices will associate with new environments and express their preferences and influence in their environments. Application development within this new paradigm, however, can be difficult, as applications require dynamic network configuration, mobility management, localization, and social and application context. To that end, I have developed Panoply, Java-based middleware that assists developers in quickly developing ubiquitous computing applications that can leverage location, social groupings, and user and application interests. Additionally, Panoply provides active device management, enabling dynamic, flexible, and simple network configuration, policy-based admission control, secure mediation of interactions, and dynamic discovery and coordination within network-based device communities. Using Panoply, I have developed a number of interesting applications, including the Smart Party—a location-aware social multimedia application, a group-driven locative media application, and a device quarantine and update management solution for mobile devices. We have shown substantial benefits can be gained from Panoply, including security benefits, scalability benefits, and also application-level benefits from device cooperation and coordination.

Chapter 1

Introduction

Mobile computing has become the norm for most computer users. We are equipped with mobile phones, laptops, digital audio players, and cameras, as well as other devices. Wherever users travel, their devices travel with them, mostly dormant digital avatars. As mobile technologies continue to permeate our homes and workplaces, we will be forced to deal with the realities of these embedded devices needing to interact, coordinate, and exert influence with and upon each other and their environments. These devices will belong to different users and different administrative domains. They will have different operating configurations, and different security and application requirements, yet they will still need to interoperate in a cohesive fashion. Problems that will arise in these environments include issues of configuration, security, scaling, and resource management, as well as coordination.

As the number of interacting devices grows, it will be beneficial for devices to group together into communities with common interests, goals, or requirements. The impetus to form these communities may derive from the devices and their applications, as they seek application benefits from grouping and coalition building. Alternatively, this impetus may be applied externally, from infrastructure seeking to reduce its load and increase the environmental carrying capacity.

The principle challenge here is vertical support for mobile applications, from the low-level networking configuration through secure connection establishment, location and social context acquisition, and finally group discovery. Existing group-based approaches tend towards application- or domain-specific solutions focusing on enabling group awareness, but in general have limited device configuration and management capabilities. Resulting systems are inflexible and incompatible, limiting the ability for cross-domain applications and interactions. The thesis of this dissertation is that new organizational and management primitives predicated upon building groups of cooperating devices will simplify both the development of pervasive computing applications as well as the management of the devices that execute them. We will show that Panoply's support for organizing and configuring mobile devices and managing their coordination improves the user experience, device security and also infrastructure scalability.

This dissertation presents Panoply, active middleware supporting dynamic, mediated communities of devices and their applications. We provide evidence that a general device management middleware supporting community-aware applications can simplify application development and yield significant advantages to at the user, device and infrastructure level in the form of improved security, application scalability, and also overall application performance through group-based coordination. Additionally, we will show that Panoply imposes reasonable amounts of system overhead upon network configuration and communication, while offering solid application performance.

1.1 Communities in Ubiquitous Computing Environments

Human interactions occur largely within the context of various communities and groups.

These groups include ephemeral location-based groups, such as one formed by a random encounter in the street, transient groups based on collaborative tasks, and formal social relationships such as the family and other human organizations. These relationships provide context, allowing us to manage, reason about, and discuss our interactions. We rely on organizational and societal policies for security, knowing that each member is scrutinized by the group. Communities can offer their members services that an individual is simply not capable of providing, such as so-called public goods. Additionally, community members draw upon each others' resources when in need. Communities collaborate, making decisions with thought to the value of the decision to the group as a whole, while also considering the impact of group choices to the individual.

Supporting a similar notion of community with respect to groups of devices offers us new ways of thinking about devices in pervasive computing environments, as well as new tools to simplify device management. For the purpose of this dissertation, we consider device communities to be mediated groups of self-selected devices operating under a common policy and cooperating to ensure continued operation and satisfaction of the community. A community-based approach towards device management can simplify the development of pervasive computing applications and assist with scaling, resource management, and security and privacy concerns.

We believe that communities are important for emerging pervasive computing applications and that middleware should support groups with dynamically changing memberships. This dissertation presents a pervasive computing middleware that abstracts the notion of communities, provides core management primitives, and provides support for developing applications. The underlying abstraction is the *Sphere of Influence*, which extends the metaphor of human communities to pervasive computing. Applications are written at the level of the sphere; this allows applications to interact in a standardized way with a single endpoint device, groups of devices, or even groups of groups. The Panoply middleware scopes resources, policy, and communication and provides a security boundary around the sphere.

The following subsections describe three pervasive computing scenarios in which Panoply simplifies development and provides substantial application benefits. These examples have been investigated in our research through real implementations, testing, and measurement. These scenarios will be used in discussions throughout the dissertation and will be explored in detail in later sections.

1.1.1 Case Study: The "Smart Party"

We envision that our mobile devices will soon participate in the same social events as we do, improving our overall user experience. They will represent the user in the digital realm, sharing preferences for music, movies, food, and environmental settings with infrastructure and also other nearby mobile devices. They will dynamically adjust to the mood of the gathering, the attendees present, and also on-going activities.

Consider the following scenario:

Bob is attending a party thrown by his friend Alice, who lives in an apartment immediately downstairs from Bob. Bob's mobile device, e.g., a PDA, associates with

Alice's network and automatically joins the community of devices at the party. As Bob moves from room to room around the party, Bob's PDA joins device sub-communities representing the current room occupants. Background music is playing and video clips are being displayed on the walls. Music is being dynamically selected from the content present on all the mobile devices in the environment through a voting process performed within each room's device community. Bob really does not enjoy the rap music being played in the living room. His PDA informs him that if he moves to the living room, he can join a minority community interested in listening to the classic jazz that Bob enjoys, enabling the community to exert more influence in that room and possibly vote in some Miles Davis.

This raises numerous technical questions. How do we find devices with which to interact at the party? How do we choose among different choices? Bob and Alice live physically adjacent to one another. How does Bob's PDA know to interact within Alice's network? How does Bob prove his authority to interact with the party? How does Bob's PDA coordinate with other devices to satisfy overall community preferences with respect to music and video? How do we limit Bob's access to Alice's network services once the party is over? Panoply must provide device community management primitives that enable Bob's PDA to:

- Configure itself for the local area network communications;
- Identify and securely join the appropriate device community for the party;

- Acquire configuration data allowing it to identify appropriate location and social communities
- Coordinate with local peers to maximize the satisfaction of communal preferences.

1.1.2 Case Study: Locative Media

Ubiquitous computing promises to tear down the traditional walls that enclose users, and let them instead leverage the real world for work and for play. One exciting application area that illustrates this principle is locative media, which incorporates physical locations and user mobility to drive a media experience. Locative media introduces virtual analogs, overlaid upon a wide range of physical environments, which lead users to explore new places and experience familiar locations in a new way. As users explore locations and the corresponding locative media, their devices interact with the environment.

Consider again our friend Bob:

It is a beautiful Sunday afternoon, and Bob is out with his wireless PDA, exploring a locative media experience in the West Los Angeles area. As Bob travels to different locations around the area, his PDA allows him to access media content keyed to his current location. In the context of a story, each user takes on a role in the story, and is associated with one or more different groups. When Bob encounters others in his physical surroundings participating in the locative media experience, their presence is detected by his PDA and affects the media that is displayed to Bob.

This example is one case of a much more general type of location-aware application. Device communities can enhance the locative media application as well as other location-based applications in many ways. They provide dynamic group-context, enabling Bob to be alerted to the nearby presence of others sharing the locative media experience. They can also assist Bob's device with issues such as localization failures, lack of local network configuration hints, or inability to access relevant local metadata.

1.1.3 Case Study: Network Security Management for Nomadic Users

Historically, devices have operated in a few well-known and established environments. As we move towards a more mobile paradigm, nomadic users travel between numerous computing environments, including their homes, their places of employment, coffee shops, libraries, and parks, as well as other locations offering network connectivity. Each of these different environments operates under wildly different conditions. Traveling between well-known and presumably safe environments and unfamiliar and potentially hostile ones poses many security challenges in mobile and pervasive computing.

Traditional networking models involve establishing a data connection between the computer and the network, and configuring the computer with a static IP address or a dynamic one via DHCP [RFC2131]. This process may involve security in the form of authentication via possession of a shared key or through EAP [RFC3748] or RADIUS [RFC2865] solutions. However, in the past, little was done after this to ensure that the newly attached device operates under the standards of the environment concerning active services, patch levels, virus scans, sensitive data, and so on. Users may continue operating machines with vulnerable and archaic configurations and take them into unsafe networks where they will quickly be compromised. Users also take already compromised devices into a home or enterprise network where they can spread infection.

Device communities can substantially improve upon security by withholding services such as Internet connectivity or local area resources until potential members participate in a mediated join process in which the device negotiates with the community for access, receiving local community policy requirements and sharing local configuration details. As part of this mediated join process, devices that desire to join a community must configure themselves appropriately to interact within the target community; this may involve updating software, disabling services, and so on. Once a member of community, devices are aware of community policy and can monitor each other's behavior and report detected misbehavior to the community at large.

Recent developments in the last few years in the area of wireless network access control echo the spirit of Panoply, using community-based policy and examination facilities to improve network security. Our work in the context of Panoply pre-dates these other efforts, and will be discussed further in related work.

1.1.4 Case Study Discussion

The above scenarios all leverage the various benefits of managed device communities to improve the user experience. These cases all possess the following important characteristics:

- *Mediated community membership:* A mediated join process implies that membership in a device community is dependent upon a successful negotiated interaction with a community representative. This is the first chance the community has to impose policy and provide configuration to the new potential member. This lets the community inform the device about restrictions or alternate communities, determine device configuration and potential vulnerabilities, and provision additional necessary configuration data.

- *Community configuration bootstrapping:* Too often, in existing ubiquitous computing systems, configuration is statically deployed. Devices are configured to operate in one or two limited environments. This is not a realistic model. In the case of the Smart Party, party attendees may have never visited the party environment, nor has anyone preconfigured his or her mobile device to localize or interact within the party network environment. The device community management infrastructure must be able to support flexible bootstrapping of device configuration, enabling devices to enter easily into new environments. Additionally, bootstrapping must require minimal user effort. Complex configuration operations frustrate and confuse users. One example of user frustration is detailed in [Pogue2007], but this sentiment is echoed by many others.
- *Device assistance and collaboration with other community members:* In many cases, an individual device may benefit from collaboration with other community members. Frequently, in real applications, mobile devices have difficulties in acquiring connectivity, properly localizing, or otherwise acquiring some desired resource. Nearby devices often can assist by providing configuration information, proxying a resource, or assisting with localization. Other times, devices will want to collaborate in order to provide better services to the users, such as finding intersections in user

preferences, or efficiently sharing network or processor resources.

1.2 Key Contributions of this Research

The key contributions of this research are the following:

1 The development of a device community model and prototype implementation

I have developed a new model of device communities called *Spheres of Influence* that incorporates automated network configuration, dynamic admission control, and membership management, as well as a publish-subscribe communications model. I have implemented this model as part of an application-oriented active middleware called *Panoply.* This research has resulted in the development of new techniques for bootstrapping device communities as well as new collaborative techniques for the mobile devices in a device community.

2 Novel applications and services that demonstrate device community principles, and provide patterns for similar types of applications

To demonstrate the middleware, I have implemented three different applications based on the described case scenarios. For all three applications, we discuss their design and implementation, and also provide key measurements to show the benefit of Panoply and Spheres of Influence within the specific application niche.

- **QED—**QED safeguards nomadic computer users and infrastructure components through quarantine, examination, and decontamination. QED demonstrates basic discovery and mediated community join. In QED, devices are required to go through a three-stage join process in order to become members of a local device community and receive services such as Internet connectivity. Our implementation of QED mandates that incoming devices stay up-to-date with respect to installed applications and services.

- **The Smart Party—**The Smart Party application demonstrates end-to-end configuration and bootstrapping of devices into a device community. Starting with minimal information, a device identifies an appropriate network-based device community, bootstraps into the appropriate location-based community and participates in a collaborative media application.
- **Locative Media**—Our locative media application demonstrates basic discovery and formation of device communities based on location, and ad hoc cooperation among devices to assist with services. In this application, groups of users tour the UCLA campus with mobile devices, exploring a science fiction narrative driven by the story's state, group interactions, and the users' locations.

1.3 Roadmap

This dissertation is laid out in the following manner. Chapter 2 discusses the issues inherent in managing device communities in ubiquitous computing, as well as the requirements for supporting these communities via active middleware. Chapter 3 describes the Panoply architecture and the prototype implementation. Chapter 4 discusses our experiences with Panoply and also provides baseline overhead measurements. In Chapters 5 through 7 we return to our case studies and look at three different applications that have been implemented on top of Panoply. Each application's design is described;

we also evaluate each application and discuss how Panoply has improved application performance and the user experience. Chapter 8 discusses related work, Chapter 9 discusses possible future directions, and Chapter 10 concludes this dissertation.

Chapter 2

Supporting Communities of Devices

The scenarios described in the introduction could be implemented as stand-alone applications, each with its own custom infrastructure supporting device grouping and group context, as other prototype pervasive computing applications have done before. The resulting applications would, however, contain many redundant infrastructure subsystems and other components that could be simplified and generalized through abstraction. In the application-specific infrastructure paradigm, devices participating in multiple applications would incur unnecessary added overhead. Additionally, this paradigm would require users to anticipate all desired interactions and pre-deploy necessary infrastructure components; we believe this is an unreasonable onus.

In our vision of the computing environments of the future, devices will come and go, seamlessly integrating with others in new environments and in new configurations in order to operate with applications and services offered in those environments. This worldview of device interaction necessitates a fluid and extensible interaction model of pervasive computing environments.

One aspect of this fluidity is the transparent formation, interaction and dissolution of communities of devices. Panoply aims to simplify interactions and application development by focusing on device communities as a principal abstraction for a ubiquitous interaction framework. This section examines different types of device communities that we are interested in supporting and analyzes their common requirements.

2.1 Group Archetypes

In order to provide a structure for this discussion, we must examine the requirements to support device communities for pervasive computing applications. At its core, a device community is a group of devices, unified by one or more communal attributes. These attributes may be derived from presence in a location, utilization of a particular communication technology, membership in a social group, participation in an application, and so on. These groups are loosely coupled, and individual devices are autonomous. These groups are an inherent part of mobile and pervasive computing, and are encountered constantly in many different forms. Groups can be used to organize shared resources and content, improve system scaling by scoping communication and interactions, and to provide valuable contextual input to user applications and services.

We can divide commonly used groups in pervasive computing into four principle group archetypes:

1. **Communication groups** are comprised of member devices that are attached to the same physical network and can, with little effort, communicate with one another. Typically, these devices discover one another via some mechanism, such as ARP messages, broadcast packets, multicast group messages, etc. Examples of this include traditional Ethernet and 802.11 infrastructure subnets, Bluetooth or IBSS-mode 802.11 cells, personal device clusters [Chetan2004, Jacobsson2005]**,** UPnP service discovery

[Golan1999], and so on. By connecting to network infrastructure, a device may implicitly be joining such a group, where the boundaries of the group are defined by the broadcast range of the medium. In addition to communication, these groups are typically used for resource and service sharing.

2. **Location groups** form around a physical place, and their membership fluctuates with the physical presence or access of individuals in and to that place. The boundary of these groups is detected through direct physical sensing of spatial coordinates, by means of passive network beacons such as 802.11 AP beacons, active probes such as those used by PlaceLab [LaMarca2005], GPS readings, etc. These sensed spatial coordinates can then be tied to a meaningful notion of place. Membership of these groups waxes and wanes based on the external phenomena of presence and connectivity. A location group may lie largely dormant until users with their mobile devices arrive in the area. There are many examples of implicit location groups in pervasive computing, ranging from home computing environments and friend-finders, e.g., Dodgeball, to transient location groups, including pervasive computing games and the popular Japanese LoveGety matchmaking application, among others.

3. **Task-oriented groups** form around a set of entities that will collaborate to accomplish individual or group tasks. These tasks may be at an application level, such as cooperating with other devices to manage environmental settings such as temperature or sound levels, or at the device level, such as coordinating to identify appropriate network configurations or optimize network utilization.

One example of a more traditional task-oriented group is the group of devices formed around a Bittorrent [Bittorrent] feed when a seed announces the existence of the group. The group grows as interested peers discover the new torrent, join, and assist others in downloading the content, and dies back as peers complete and disconnect from the torrent. Typically, some members continue to seed the torrent until there is no longer any interest and the torrent eventually dies off. Despite coming from the non-pervasive world, BitTorrent is an excellent example of the discover-interact-dissolve model we have proposed. The BitTorrent model has been extended to the pervasive computing by [Jung2007], which focuses on allowing local-area Bluetooth-enabled devices to collaborate in order to collectively download large media files.

We, along with others [Hoebeke2006], believe that these types of dynamically formed task-driven groups will be increasingly useful and important as more and more mobile applications are deployed.

4. **Social groups** form around a set of entities with common interests or relationships, typically based on user relationships, interests and applications. Social groups may be declaration-based, where members declare their association with the peer group, such as within a social network graph, or via a user group profile [Wang2005]. Alternately, a social group may be participation-based, implicitly created through interactions within a peer group, such as a school class dividing into project groups.

These classifications are not mutually exclusive, nor do they necessarily represent the whole space of interesting groups. A single application may easily fall into two or more of these categories. This overlap is inherent in pervasive computing applications as these different group archetypes represent specific types of functionality. By combining different functionalities, we enable new types of applications. Examples include location-aware instant messaging [Eisenstadt2002], and Google's Dodgeball service **[**Dodgeball**]**, wherein both integrate social community context with location group context, and personal device networks, such as [Jacobsson2005], which integrate social and communication or task context.

2.2 Community Building

As described above, a device community is a group of devices, unified by one or more common attributes. However, community membership extends beyond the binary of "in" and "out." By mimicking the behavior of human communities through cooperation, charitable assistance and policymaking, applications can improve the individual user experience.

The Panoply application paradigm encourages community-driven applications and incorporates community-oriented features to improve the overall experience for users. User devices and applications work together to assist in furthering the goals of the group. Community behavior can be encouraged through device collaboration, community aid, and community regulations.

- *Collaboration*: Community members actively work together to improve the shared experience. This can include cooperating to identify intersecting user preferences, synchronization or turn-taking to allow equitable and safe resource sharing, clustering to reduce environmental overhead, task division, etc.

- *Community Aid*: Community members assist one another in times of need. This could include providing configuration hints for lost or misconfigured devices, providing a local cache of some data, or proxying for devices that are unable to connect to the infrastructure.
- *Community Regulations*: Each device community imposes a reasonable set of basic regulations, or requirements, upon incoming supplicants and current members. These regulations serve to guide behavior within and throughout the device community, and may also limit or encourage certain types of behaviors and require certain forms of device authentication. They may even require basic system examination in order to protect the community at large.

2.3 Functional Requirements for Supporting Communities of Devices

By drawing out commonalities in the different archetypes we can identify a core set of high-level abstractions that are necessary to support device communities and community-aware applications, as well as manage configuration. This allows the development of an active middleware to manage memberships and configuration, and tools to support applications operating in these contexts. We have identified several core requirements to support communities, including advertisement and discovery, configuration, policy mediation, and communication support.

2.3.1 Advertisement and Discovery

Once a device elects to form a group, or decides it is interested in joining a group, it needs a mechanism for alerting others to the presence of the group. Traditional human groups may advertise through word of mouth, or send out mailers. Advertisements may also take the form of a posted flyer or sign that advertises a group's existence. These advertisements may be a description of a group that includes pertinent details such as the purpose of the group, membership requirements, and contact information. In pervasive computing environments, there are analogs for all of these different forms of advertisements, and different groups may require different advertisement mechanisms.

Communication groups require beacons or advertisements, such as 802.11 infrastructure beacons or ad hoc cell advertisements, or multicast group announcements. Other groups may advertise via "electronic flyers," effectively a published mapping from some interest, task, or location to a network address that will allow contact with and potential membership in the group.

For every advertisement component, we also require a corresponding discovery component. Beaconing advertisements necessitate beacon listeners capable of capturing and decoding community advertisements. Location-aware discovery mechanisms need to be able to sense environmental characteristics and, using advertisement information, transform those characteristics into appropriate community contact details.

2.3.2 Automated Configuration and Bootstrapping

An important part of the discovery system is the ability to automatically configure and bootstrap the local device into appropriate device communities. Devices will need to be able to configure and reconfigure themselves in order to sense and join device communities in which they wish to participate. Communication groups may require MAC-layer reconfiguration such as selection of wireless SSIDs, or proper selection of appropriate IP addresses. Devices participating in location-based groups may also require location bootstrapping to acquire current localization maps for pertinent regions. Automated configuration is particularly important in order to achieve real ubiquity of mobile computing. With the penetration of Wi-Fi into user homes, users are facing system administration nightmares from inconsistent and incompatible system services; home network problems include competing instances of DHCP, multiple levels of NAT, automatic roaming of wireless devices to incorrect networks, and improperly configured and insecure printers and scanners offering their services to any interested passerby. Expecting users to perform complex and arcane configuration tasks to join and configure device communities is unreasonable. Device community configuration needs to be streamlined and substantially automated in order to reduce confusion and frustration in end users.

2.3.3 Security and Policy

Security and policy are essential to device communities—together they dictate and enforce the rules that regulate device behavior. There are two principal design challenges that must be addressed. First, device communities require a flexible model of secure association to support dynamic configurations. Supplicant devices and the target community may possess no a priori knowledge of one another. Flexible mechanisms are thus necessary to allow the formation of a secure association between the supplicant device and the community. Second, different communities will have different policies regarding membership and interactions. Flexible, policy-based join mediation and interaction facilities are necessary to support a wide-range of applications and different types of communities.

2.3.4 Communication Facilities

Clearly, devices participating in a community must be able to communicate and interact. Additionally, applications need to be able to discriminate among the multiple device communities that a device may participate in and selectively communicate with those that are most relevant. In many group-based applications, nodes communicate through application-specific protocols, making interaction between and among different types of applications and communities difficult. A standardized communication mechanism would allow interactions between different applications and heterogeneous device communities, enabling new applications as well as new ways of connecting old applications.

Chapter 3

Active and Community-Based Middleware for Pervasive Computing

With our community requirements from Chapter 2 in mind, we turn to a high-level design overview of our research framework to support general device communities for pervasive computing applications. Section 3.1 will provide an overview of the operation of Panoply and the basic services it provides to developers and applications. Section 3.2 will go deeper into the Panoply architecture, discussing the various systems that compose Panoply, and their interoperation.

3.1 Panoply Operational Overview

The different communities described previously are used for a variety of purposes, but they have common requirements that we believe demand the development of a generalized software layer. Additionally, this supporting layer may enable new types of applications that build upon interactions among heterogeneous device communities. This software, named Panoply, is an active middleware framework.

3.1.1 An Active Middleware Approach to Pervasive Computing

Traditional middleware is a software layer used by programmers to mediate interactions between distributed computing components. Middleware sits between the application and the operating system, and manages interactions between disparate applications across heterogeneous platforms. Typically, middleware is accessed via application program interface (API) calls. Middleware simplifies the software developer's task by abstracting otherwise complex tasks into a small set of API calls. Well-known examples of middleware technologies include CORBA [OMG], and Microsoft's DCOM.

Active middleware builds upon traditional middleware approaches by taking a persistent, "active" role in managing system resources and behavior on behalf of applications and user preferences. Active middleware can act on its own in response to external events, and is not limited to taking actions in response to API calls. Active middleware is thus more reactive and agile, able to take actions on behalf of the user without necessarily requiring specific application control. This model is powerful and very appropriate for pervasive computing applications, which need to react to changes in user location and context. Additionally, this approach consolidates the complexity of managing connections, interactions and policy into a central system component which can remain active and aware of user preferences and device configuration even when applications may not be active.

3.1.2 The Sphere of Influence

At its core, Panoply manages device relationships and interactions with the environment. The central abstraction behind these relationships is the *Sphere of Influence* [Eustice2003a]. Our task of abstracting device communities requires a general community object with a standardized interface, as well as a reflection facility to allow other devices and applications to access and inspect the device community. In the spheres of influence model, devices and device communities are represented as software entities called **spheres.** The sphere is a network construct, representing a community of interacting devices and the context within which they interact. The sphere also serves as a common representation for application end-points in Panoply. The sphere model ensures a consistent interface at different levels of granularity, from the single device to complex groupings. This allows applications to interact with groups of devices as easily as with a single device.

Formally, a sphere represents a single device, or, recursively, a structured group of spheres. Every device is represented as a sphere. More complex spheres are formed through aggregation with an existing sphere. Other devices (supplicants) join together with an existing sphere in a structured manner. The hosting sphere is called the *sphere leader*, and it serves as the coordinator for the aggregate sphere. We refer to the non-leader participants in the sphere as members. Additionally, we use parent-child notation to describe the relationships: e.g., if sphere A is a member of sphere B, A is a *child* of B, and B is a *parent* of A. Additionally, if two spheres have a common parent, they are said to be *siblings.*

Principally, we divide the space of spheres into five categories, based on function and operating behavior. Table 1 below summaries these categories. These divisions are primarily for categorization, since a given sphere may exhibit behaviors from any of these categories.

Sphere Class	**Discovery Mechanism**	**Principle Functionality**
Device	*n/a*	Represent a device
Communication	Network beacons	Represent network infrastructure
Location	Localization system	Manage a physical location
Social	Static configuration, or advertised social identifiers	Serves as a known point of contact for applications
Interest/Task	Advertised identifiers via pre-existing channels	Coordinate and organize participants in a task

Table 1. Summary of Sphere Categories

Device Spheres: The device sphere is elemental—it consists of a single device. This sphere represents the device and is the point of contact in the Panoply framework for all interactions with the outside world. The device sphere discovers, joins, and connects with other spheres in order to access network infrastructure and interact with relevant services, as well as participate in community applications.

Communication Spheres: Spheres of this type are detected via network proximity, and members of a communication sphere typically share a common network infrastructure. This type of sphere requires a discovery component that recognizes the communication sphere's presence by detecting a network identifier such as a beacon, the use of a specified gateway, a specified IP range, etc. An *ad hoc* sphere is a special instance of a communication sphere. If a device sphere cannot find or otherwise access the available infrastructure, it may form and advertise an ad hoc communication sphere. Ad hoc communication spheres will be discussed fully later in the architecture section, and their use will be illustrated in the Locative Media application described in chapter 6.

Location Spheres: A *location sphere* is associated with a describable physical locale, such as a room, building, park bench, etc. Complex locations may be structured as a hierarchical arrangement of spheres, such as a set of *room* spheres in a *building* sphere. Member spheres require localization context in order to identify relevant location spheres. Given some localization data—e.g., a localization database of some form, a supplicant sphere determines if a mapping exists between some sensor input (GPS, UWB, 802.11, etc.) and location spheres described in the localization database. This mapping is not necessarily one-to-one, since multiple logical locations may be overlaid on a single physical location.

It is important that the localization database be dynamically extensible in order to handle new environments that the user may visit, or new applications that may be used on the mobile device.

Social Spheres: A social sphere is a device community whose members participate in some common application or service; the social sphere is used as a shared "social" communication medium for those applications. It is typically not dynamically discovered but rather preconfigured by a user application. The user application may request that the device sphere maintain a connection to the social sphere whenever network connectivity is available, or the user application is active, etc. Examples of these communities include a coordination sphere used by a social networking application, a communication server for a multi-player game, or a master control and communication center for all of the user's personal devices.

Interest-Based Spheres: Interest-based spheres are highly dynamic and typically transient spheres that form as needed around a member group with some shared interest or task, in order to organize some activity. An interest-based sphere is created in an ad hoc manner from spheres already participating in some larger community. Spheres may advertise their intent to form an interest-based community, and other interested spheres may discover and join the new interest-based sphere. Examples of interest-based spheres might include the formation of preference coalitions and coordination of preferences within a smart environment, task groups to divide up a problem among multiple interested devices, etc.

Basic Sphere Operation

A given sphere typically interacts in two different modes: the first is as a **supplicant sphere**, seeking out other spheres in which to participate. The second mode is that of a **leader sphere**, where the sphere receives members for interaction purposes.

3.1.2.1 Supplicant Sphere Operation

Basic supplicant sphere operation consists of five principle phases—solicitation, discovery, configuration, join, and interaction. A sphere may be participating in multiple supplicant phases simultaneously with other different spheres. Figure 1 shows the basic state diagram that governs sphere behavior, and illustrates the relationships between the different phases. This state diagram depicts the basic functionality of the supplicant phase of a sphere; the actual Panoply implementation is more complex and handles edge cases such as a multiple on-going joins, intermediate join failures, etc.

Without loss of generality, the discussion of sphere operation will be primarily from the perspective of a device sphere, as they typically exercise this functionality frequently; other types of spheres follow the same behavior when interacting as supplicants.

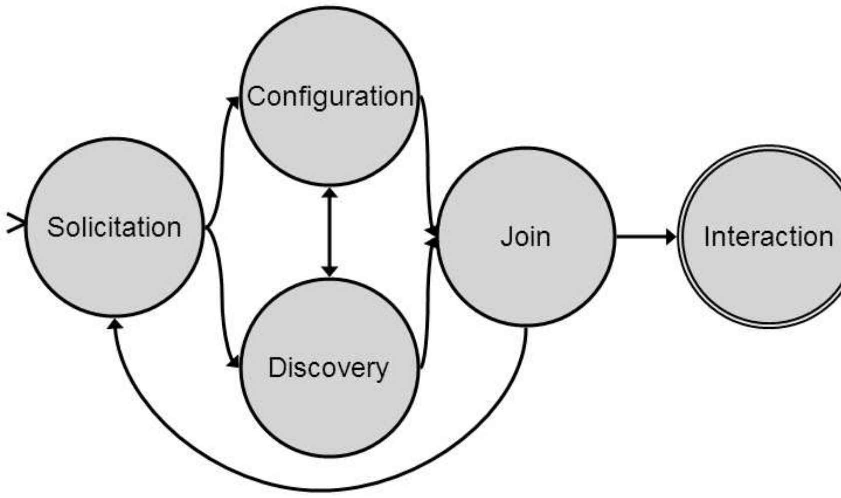


Figure 1. Basic supplicant sphere state machine

Solicitation Phase: *Determine the need for interaction*

Initially, the sphere requires that some component, either a Panoply system component or a Panoply application, issue a request to initiate the supplicant state machine. This request may specify the nature of the sphere, relevant network configuration, or other hints to assist with the process. Given such a request, the sphere enters either the discovery phase or the configuration phase.

Discovery Phase: *Discover relevant and desirable device communities*

In the discovery phase, the supplicant sphere attempts to identify relevant communities that satisfy the solicitation. This may involve attempting to discover a local communication sphere, localization to identify an appropriate location sphere, or identifying interested peers with which to coordinate. Once an interesting sphere has been detected, the supplicant either moves into the configuration phase for additional configuration (if necessary), or transitions straight into the join phase.

Configuration Phase: *Configure local device as necessary to attempt to join a sphere or enter the discovery phase*

The configuration phase involves reconfiguring the local sphere based on hints provided in the solicitation phase or discovered configuration details from the discovery phase. Configuration changes include layer 2 and layer 3 network parameters, localization database updates, application launching, etc. After completing configuration (or reconfiguration), the supplicant either moves into the discovery phase, or, if it already have discovered an interesting and relevant peer, the sphere moves on to the join phase.

Join Phase: *Join and negotiate with device community*

In the join phase, the local sphere contacts a discovered sphere and attempts to negotiate the terms of sphere membership. The join may fail due to failure to negotiate acceptable terms of service, or due to network failure. Additionally, to support legacy networks, e.g., infrastructure networks without communication spheres, the sphere may enter the join phase with a null target sphere. In either case, the sphere transitions back to the solicitation phase, where it waits for further input.

Interaction Phase: *Interact within device community, subject to policy and authorization provided by community*

The final phase of the supplicant is the interaction phase. This phase is conceptually fairly straightforward; however, it encompasses nearly all application interactions. The supplicant sphere maintains a connection to the remote sphere and interacts within the context of that sphere. Communication is handled via a publish-subscribe event model. Applications can interact with other applications (both local and remote) as well as with Panoply system components. The details of the interaction phase and communication support will be discussed in more detail in both the services and architecture portions of this chapter.

We will now walk through two examples to illustrate the supplicant operation.

Discovering a Location Sphere with Panoply

Consider our friend Bob, who is already participating in a local communication sphere for UCLA's Computer Science Department with his laptop. How can Bob identify and join the appropriate location sphere that will allow him to access other devices and resources in his immediate vicinity? Initially, Bob's laptop device sphere is participating in the local infrastructure communication sphere. A local map service has indicated to Bob's laptop that a localization map for the environment is available. Based on this input, Bob's office management application triggers a request to join an appropriate location sphere. Bob's device sphere enters configuration mode, where it requests, receives, and processes a localization map provided by the local map service. The device sphere then enters discovery mode, where it processes environmental input and compares it to the localization database. If it can identify its location and that location maps to a known location sphere, the laptop device sphere then attempts to join the location sphere and interact with local services and other local devices.

This example assumed Bob’s laptop was already participating in a communication sphere—how did Bob’s laptop bootstrap into that sphere?

Acquiring a Communication Sphere with Panoply

Bob’s laptop is running Panoply; initially his device sphere is in the solicitation phase, waiting for input. Bob’s laptop does not have connectivity, so a network management component issues a request for a connection to a wireless communication sphere. The solicitation specified wireless, so the device sphere configures the wireless interface to monitor for 802.11 beacons, and then enters the discovery mode. In the discovery phase, discovered 802.11 beacons are compared against a list of community certificates that map relevant 802.11 MAC and SSID pairs to a network configuration and network addresses for the corresponding communication spheres. When the device sphere matches a discovered beacon against an entry in the list, it re-enters the configuration phase, activating the appropriate configuration by associating with the indicated network and either using a pre-configured static IP address or acquiring a DHCP address.

Next, the device begins the join phase with the network address of the detected communication sphere. After a successful negotiation, the device sphere transits into the interaction phase, where it can communicate and interact with other local spheres, and utilize local services such as a network gateway or a printer. From a very high level, this procedure is similar to the automatic association techniques used by Microsoft Windows Wireless Zero Configuration [MSWZC], which automatically manages wireless associations by pattern-matching detected 802.11 SSIDs against a prioritized list, and auto-selecting the matching network with the highest priority. The Panoply approach is similar in kind, but is much more general, including the use of time-limited configuration profiles which are only active during a prescribed time period, the use of an observed AP's MAC,SSID pair to trigger an association with any specified AP, and the use of both SSID and MAC information to prevent over-aggressive (and incorrect) associations. Additionally, we support the ability to tie an application or set of applications to a network and automatically launch the application(s) as appropriate, and of course, the ability to tie a network profile to a specific device community to be joined upon activation.

If no Panoply communication sphere is available, the same procedure can be used to connect to a non-Panoply network, terminating after IP address acquisition.

3.1.2.2 Leader Sphere Operation

In addition to the supplicant mode of operation, some spheres also act in a leadership role. In the leader sphere mode of operation, spheres configure themselves and advertise for peers. As peers attempt to join, the leader sphere can then choose to accept the incoming peer and allow it to interact. The basic state diagram is depicted in Figure 2. The leader sphere states are similar to the supplicant states, with a few exceptions.

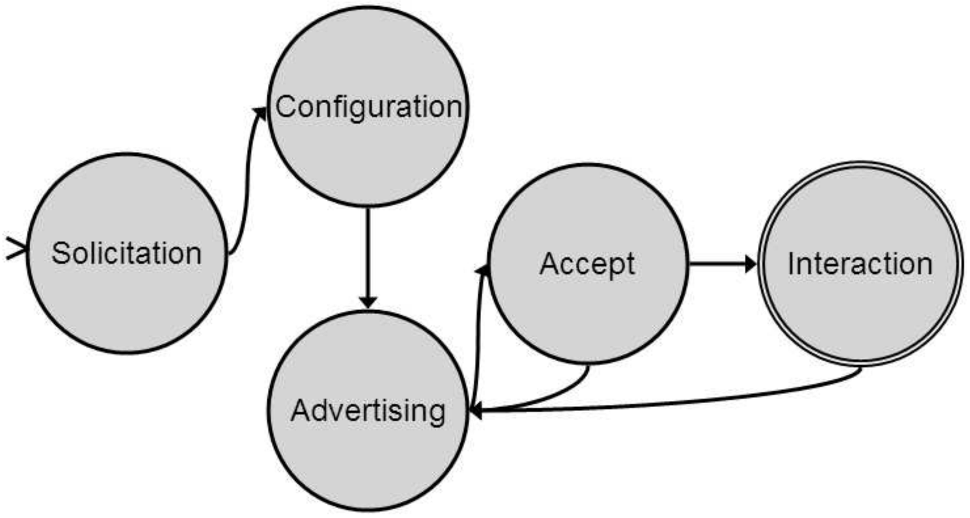


Figure 2. Basic leader sphere state machine

Solicitation Phase: *Determine the need for initializing the leadership process*

Initially, the sphere requires that some component, either a Panoply system component or a Panoply application, initiate the leader state machine. This includes specifying the sphere configuration for the new sphere, including the sphere name, type, operating policy, relevant applications, and a set of advertising components that the sphere will use to advertise its presence. Given the appropriate initialization data, the sphere enters the configuration phase.

Configuration Phase: *Configure and initialize sphere components*

In the configuration phase, the leader sphere configures and activates necessary advertising components and initializes them. At this point, any applications associated with the leader sphere mode, as specified in the solicitation phase, may also be launched.

Advertising Phase: *Advertise sphere presence and wait for connections*

Once the advertising components have been initialized, the leader sphere is actively advertising its presence. This may take different forms, including advertising via a network beacon, advertising a localization map, etc. These parallel the forms of discovery. It may also not involve any active components; for instance, in the case of a social sphere with a known network interface, advertising is implicit in the application configuration. Once a supplicant sphere contacts the leader sphere, the leader sphere begins the accept phase. In terms of implementation, the advertising phase is ongoing, while the sphere forks off a new accept process to handle the incoming supplicant sphere.

Accept Phase: *Negotiate with the incoming supplicant for access rights*

After the leader sphere has been contacted by a supplicant, both spheres participate in a negotiation process to determine if the supplicant sphere can and will become a local sphere member. Negotiation is handled by an integrated policy management facility that determines the acceptability of the peer based on a series of exchanges of information representing information about the spheres, as well as attestations and promises regarding its behavior. If the negotiation process completes successfully, the supplicant sphere is promoted to a full sphere member and then can interact within the context of its new parent sphere. If the spheres cannot negotiate their terms of interaction, the accept phase terminates and the network connection is severed.

Interaction Phase: *Manage communications and interactions for members*

The leader sphere manages the interactions of the aggregate sphere that is comprised of it and the other sphere members. As discussed above, communication is handled through a publish-subscribe event system. Members submit event subscriptions to the leader sphere, which manages subscriptions and mediates event flow, disseminating events based on subscriptions, local policy, and scoping rules.

High-Level Functionality	**Core Services Provided**
Event System	Interest-based event dissemination Subscription management Event scoping model and scope enforcement
Application Interface	Supports application integration and communication with the local sphere
Security Services	Sphere vouchers Sphere introduction and enrollment Voucher-based state attestations
Community & Configuration Services	Managed network configuration Extensible discovery and advertising system Localization & localization management Dynamic leadership management Connection management Time discontinuity awareness
Policy Services	Membership mediation and negotiation Event mediation Policy-based event-condition triggers

Table 2. Summary of Panoply middleware services

3.1.3 Panoply Services Overview

Panoply provides a number of services that can be used by Panoply system components and also by Panoply applications. Services are typically accessed through the Panoply event system, either through creating long-lived event subscriptions (for flow-based services, such as location updates, discovery messages, or membership changes), or through published control events and the corresponding responses. Table 2 summarizes the core Panoply services that are provided. These services have been fully implemented in the existing software. Each system service will be discussed in detail below.

3.1.3.1 Panoply Event System

All component communication within Panoply occurs in the context of an integrated publish-subscribe framework. Publish-subscribe, or “pub-sub,” frameworks decouple senders and receivers, and allow for easy one-to-one, one-to-many, many-to-many, and many-to-one communication scenarios. This is appropriate to the highly dynamic and varied pervasive computing scenarios that we and others [Jacobsen2001, Cugola2001, Yixin2006] envision. Additionally, the general nature of an event recipient in the pub-sub model meshes well with the notion of the sphere as an end-point, either as an elemental device sphere, or as an aggregate of many devices.

Within Panoply, every sphere possesses its own event processing system and can act as an event subscription registrar; this differs from typical pub-sub systems that separate the registrar and recipient roles into separate systems. Sphere components and applications express their interest in a set of events by registering with their local sphere for the desired event types. Any entity interested in receiving events must implement the *EventReceiver* interface. This interface specifies several required methods that components must implement—most important among these is the *queueEvent(Event)* method that will be called to send events to the component. The *EventProcessor***,** a core

Panoply component, manages event subscriptions and event flow. When an event enters the EventProcessor, it is compared to the subscription database. If any local sphere components, applications, or related spheres have a subscription for the specified event type, a copy of the event is pushed into the corresponding event queue. Reliability is not guaranteed or even implied. If an application requires reliability, it currently must be implemented at the application layer.

Event subscriptions, also known in our system as *event interest*, are described using Panoply's flexible event type model. System events are typed by primary type and sub-type. Additionally, applications can define their own event types, allowing for nearly unlimited extensibility. Primary types and subtypes can be composed, allowing for a wide range of possible system events. Principal event types include *discovery, membership, location, policy, cache, heartbeat, management*, etc. These events are typically generated by core Spheres of Influence components and used by applications to customize their behavior. *Appendix II: Panoply Events* lists all event types and sub-types, and indicates how they are used.

Application event interest is propagated between related spheres, to the extent that policy permits. Control events are not automatically propagated between spheres; this is done in the interest of security, and also because most control events would be irrelevant in the context of a different sphere. When a sphere becomes part of a larger sphere, the child and parent cross-register their event interest as necessary.

In addition to interest-based event routing, Panoply also supports a *directed* event type that delivers an event to a specific sphere via a specified path, in a manner similar to source routing [Johnson1996]. If the path is incorrect or incomplete, the event is dropped.

Scoping Event Flow

As devices participate in larger and increasingly complex group structures representing different locations, applications, and social groups, it becomes necessary to control event flow and restrict exposure of events to relevant spheres through scoping. Panoply allows each event to have an attached scope that limits its propagation. Limiting the visibility of events offers a simple yet powerful tool to ubiquitous computing applications, allowing them to customize the dissemination of their events. Additionally, event scoping improves scalability by reducing event passing overhead that would otherwise be incurred with potentially no gain in functionality.

Currently, Panoply offers four customizable event scopes:

- a hop-count-based *time-to-live (TTL) scope;*
- a *single-path scope*, which delivers only to the first interested recipient encountered;
- a *type-following scope,* which propagates to all interested recipients of a specified sphere type, and recursively to all their relations of the same type, and so on;
- a *type-seeking scope,* which propagates to spheres of a specified sphere type only after it has transited through spheres of another specified sphere type.

Figure 3 through Figure 8 each depict a graph of connected spheres, where the sphere type is represented by the letters D (device), L (location), and I (interest). This graph is representative of a location-situated application in which multiple devices are operating in a hierarchically decomposable location, and also participate in an interest-based or socially-based device community, such as our locative media application, or the Smart Party. In each graph, every sphere is interested in a single application event type **bar**. The arrow in each graph represents the insertion of an event of type **bar** by an application operating in the context of the indicated sphere, and the shaded nodes represent the spheres that accept the event for local handling by a subscribed event receiver.

Our first example, Figure 3, illustrates the event propagation absent a specified scope. In this example, the event is delivered to every interested peer. This effectively floods the network of interested peers with the event. Clearly, this can lead to problems when there are many interested peers.

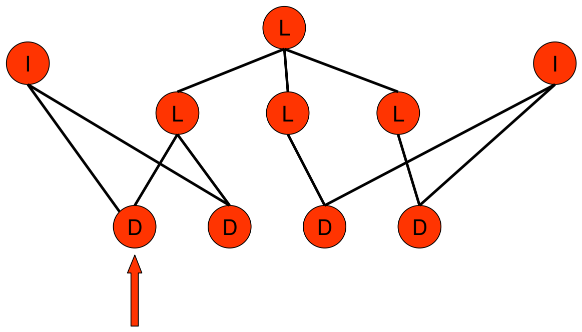


Figure 3. Unscoped event dissemination

Figure 4 depicts TTL-based event scoping. The injected event specifies a *TTL event scope* <= 2 which limits the propagation of the event to connected spheres within a dissemination radius of two hops.

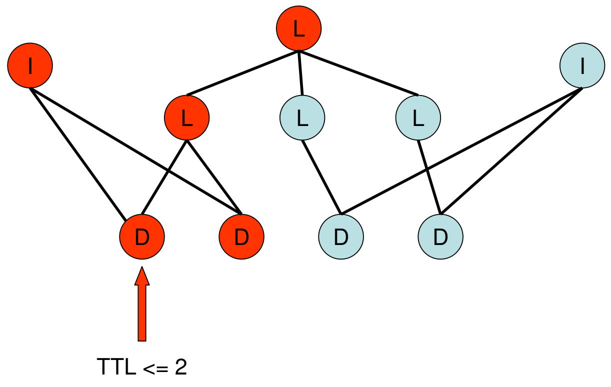


Figure 4. TTL event scoping

The next figure, Figure 5, demonstrates *single-path event scoping* where the event scope restricts propagation to a single path, and terminates when it finds a recipient for the event. For purposes of this example, we assume that the injection site does not deliver the event to a local component; if this assumption did not hold, the event would be delivered to that component and delivery would immediately terminate.

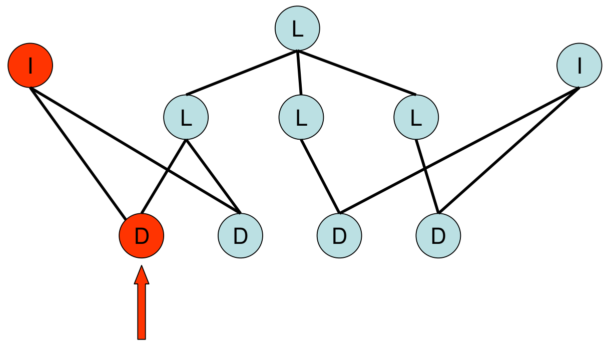


Figure 5. Single-path event scoping

This type of scoping ensures that only one event receiver receives the given event; however, the event scope does not provide a guarantee regarding which path would be followed. The event propagation depicted in Figure 6 is equally valid.

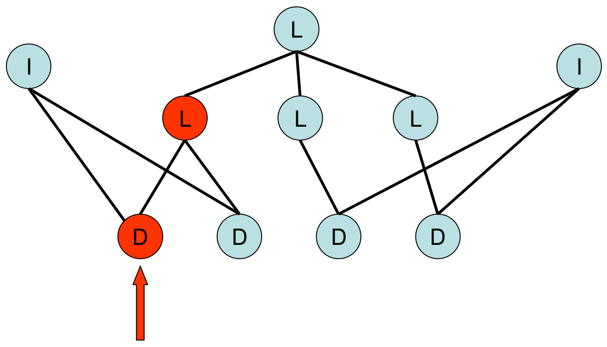


Figure 6. Single-path event scoping (alternate)

The next figure, Figure 7, depicts *type-following event scoping*. This type of event scoping allows Panoply events to limit their propagation to spheres of a specific class, such as device, location, interest, social, etc., where the sphere type is encoded as a sphere state property maintained on each sphere, and is accessible to related peers via an internal reflection facility. In this example, a **bar** event is published with a scope limiting its propagation to location spheres. As it travels, intermediate event processors only forward the event to subscribed receivers of the appropriate type.

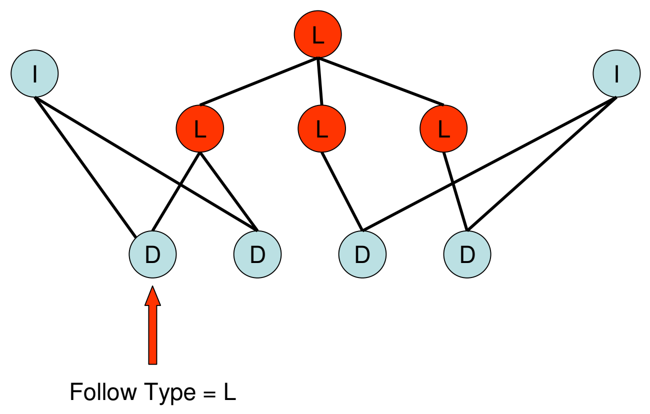


Figure 7. Type-Following Event Scoping

The *type-seeking event scope*, illustrated in Figure 8, is similar to the type-following scope, except the scope specifies a target sphere type, the intended event recipients, as well as a transit set. In this figure, the event transits through the location spheres, and is delivered for handling at the device spheres. The event is not delivered to other applications at the originating sphere, as it has not yet transited through a sphere of a type specified in the transit set.

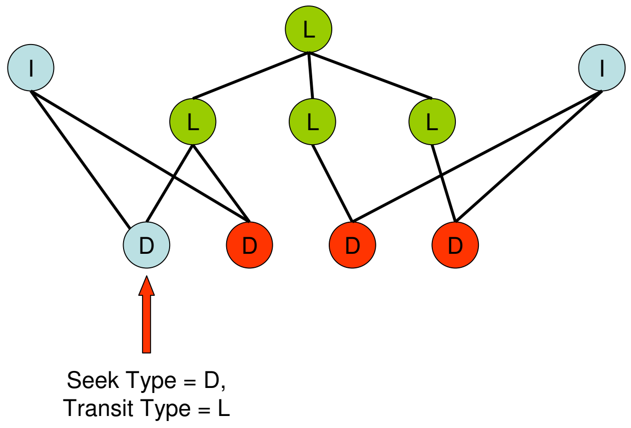


Figure 8. Type-Seeking Event Scoping

Application policy, enforced through the policy manager's hooks in the SphereAppChannel, can limit acceptable scoping. For instance, default policy should not allow unscoped event dissemination, in order to prevent network flooding.

Further discussion of event scoping and the manner in which Panoply applications utilize event scopes will be included in the discussions of the various Panoply applications (see Chapters 4 through 7).

3.1.3.2 Panoply Application Interface

Applications interface with Panoply by extending the *SphereApp* class. The SphereApp base class handles establishing and maintaining a connection to the local device sphere. Additionally, the base class provides queuing methods that allow the application to register for events, access the incoming event queue, and send its own events.

Applications can send and receive most types of Panoply system events, as well as custom-typed application events; the only requirement is that events and all event components must be serializable. Events of particular importance to applications include membership events, location events, and state events.

Membership Events: Membership events—in the form of membership updates, membership query events, and membership response events—are used to update components with respect to current sphere members. An application can register for membership events, and then be alerted when members come and go from the parent sphere. When an application starts, it is typical for it to issue a *MembershipQueryEvent*, which requests that its hosting sphere respond with a copy of the current membership table; this includes both parents and children. This information can then be used as path input, allowing the application to address *DirectedEvent* messages to parents or children of the current sphere.

Location Events: Applications that use location context typically subscribe for location events. These events provide location updates to the application. Location updates are provided at a configurable interval and come in the form of REFRESH as well as UPDATE events.

State Events: State events allow the application to query a sphere's **sphere state**. This allows the application to inspect the sphere's configuration and properties. These may include the sphere type, sphere component operating parameters, and so on.

3.1.3.3 Security Services

Panoply has a number of integrated security services that simplify application interactions and system operation. A secure data structure, the *voucher,* is used to provide security in a number of Panoply operations including enrollment (acquiring permission to join a sphere) and sphere join (the actual network operation of connecting to a sphere). They are also used to provide secure configuration context for device spheres, assisting them with proper selection of networks, location spheres, and social spheres.

Panoply Vouchers

Vouchers are a general cryptographic data structure, somewhat similar to a cryptographic certificate. Traditional certificates, such as X.509 certificates [X509-AM] are typically used to prove one's identity by leveraging a well-known and trusted certificate authority, or, in the manner of SPKI certificates, allow a single entity to authorize some action or grant a capability to another entity.

Panoply vouchers generalize this model. Beyond providing standard cryptographic information, such as keys and signatures necessary for authentication purposes, vouchers allow one or more spheres to delegate rights and specify arbitrary properties about the vouchee. Additionally, vouchers are used by third parties to associate observable phenomena with configuration profiles, such as spheres to join, or applications and Panoply components to launch. The voucher data structure is detailed in *Appendix I: Panoply Vouchers,* and is described more fully in [Karuza05].

Sphere Introduction and Enrollment

Vouchers can be particularly useful to simplify device introduction and enrollment. Introduction and enrollment are techniques where a new device can be safely and securely added to a network or device community, typically through the use of an out-of-band or side-band technique [Stajano1999], such as an infrared exchange [Balfanz2004] visual authentication using barcodes [McCune2005], an acoustic exchange

[Goodrich2005], and even a shared acceleration signature [Mayrhofer2007.] To enable secure sphere enrollment within Panoply, we have implemented a secure time- and space-limited enrollment protocol that uses a USB flash drive to provision supplicant devices with cryptographic invitations. Users entering a secure environment may use a USB flash drive to acquire and provision their mobile device with a voucher. This procedure safely and securely allows the visiting device to be added to the environment, and also provides the environment with some information about the physical presence of users and their devices. Full details of this mechanism are in [Glazer2007].

In many situations it may be inefficient or impossible to submit to an explicit introduction procedure in every visited network. For example, systems with distributed or non-centralized access control, networks with multiple administrative domains, and other similar networks have characteristics that would impose an undue burden upon network participants if out-of-band or side-band enrollment was required at every access point.

In these cases, it is desirable to use vouchers to streamline the introduction process. Vouchers contain material detailing the principal's identity (who is being vouched for), the vouching identity (who is doing the vouching), the introduction method used, time of introduction, role of the principal in the vouching network, duration in the network, etc. Vouchers can expedite enrollment in a network, given that a trust relationship between the new network and an appropriate vouching entity or entities exists or can be formed. Basic considerations, such as the time of the original introduction, location of the vouching network, etc., must be taken into account. The following example illustrates this use of a voucher.

Joe visits network A and participates in introduction and enrollment procedures in that network. Shortly thereafter, Joe visits network B, which is physically close to network A but does not possess a shared enrollment database. Instead of requiring Joe to go through another introduction and enrollment procedure, Joe gives network B a voucher from network A that attests that Joe went through introduction in A at time *t*. B trusts A to vouch for the introduction, given that the time of introduction is within some window *w,* determined based on B's proximity to A. As long as the current time $t' < t + w$, B accepts the voucher in lieu of introduction, and B is allowed to enroll.

This model gives B a great deal of flexibility when admitting Joe's device. The B network can admit Joe when the network is not highly utilized, and reject Joe when it is. In addition, B can reject Joe if the remaining lifetime of Joe's voucher from A is not sufficiently long.

Voucher-Based State Attestations

Another valuable use of vouchers within Panoply is *sphere state attestation*. State attestation occurs when one sphere requests that another sphere provide proof of some property. Upon request, an attesting sphere will generate a voucher for the requestor that vouches for the specified state, assuming such a request is compatible with local policy. Sphere membership attestation is a common example of this, where a sphere may request that a peer prove membership in another sphere, perhaps to access controlled resources or for admittance. This use of a voucher is similar to that of a membership card, or a student identification card that attests to the bearer's membership in a third-party institution.

3.1.3.4 Community and Configuration Management

Community and configuration management is another important service provided by Panoply. This service provides active management of a device's participation in Spheres of Influence, and the corresponding network and device configurations. This functionality depends upon a number of important subsystems which will now be described, including network configuration management, a discovery and advertising framework, a dynamically extensible localization facility, and a dynamic leadership management protocol for interest-based spheres. Additionally, we will describe Panoply's connection management model, as well as Panoply's time discontinuity detection facility.

Network Management and Configuration

Panoply provides programmatic access to the Linux networking stack, however for security reasons this functionality is not directly exposed to applications. Network configuration is handled by several key Panoply subsystems. The most important of these is the network manager, which manages a database of vouchers that are used to trigger configuration changes and to access other spheres. These vouchers may be provided by installed applications, through enrollment in other spheres, through email, etc.

When appropriate phenomena are observed—802.11 beacons, sensed locations, etc.—the network manager activates the matching voucher, either automatically or after prompting the user, depending on whether or not the observations match one or multiple valid vouchers. Activating the voucher may cause a network to be joined, Panoply components to be launched, or a sphere to be joined, or any combination of these. This general configuration mechanism allows the device sphere to bootstrap into the communication sphere, and then load or launch necessary components to detect and join location and/or social spheres as necessary, as illustrated by the examples in Section 3.1.2.1 *Supplicant Sphere Operation*.

Discovery and Advertising in Panoply

Panoply includes an extensible discovery and advertising framework. Component managers dynamically load and instantiate user- or application-specified discovery and advertising components. These components are registered with the corresponding manager that maintains control over the component.

The current prototype includes a number of discovery and advertising components, including modules for handling 802.11 beacons, UDP beacons, and multicast beacons.

802.11 Beacon Handling: The WiFiDiscovery component provides a basic 802.11-beacon monitoring facility, leveraging the basic PlaceLab 802.11 WiFiSpotter interface [LaMarca2005] which interacts with the Linux Wireless Extensions [Tourrilhes1997.] At configurable intervals, the 802.11 interface broadcasts 802.11 probe requests on all legal channels. Responses to the probe requests are recorded, and the results are returned via JNI to the WiFiDiscovery discovery module. This data is then fed into Panoply as a discovery event; typical subscribers for this event stream would include the Network Manager, which uses the event stream to look for possible triggers for joining communication spheres, and also the Semantic Location Mapper, which uses the event stream to perform localization.

In addition to the WiFiDiscovery component, we have also implemented a WiFiTraceDiscovery module that can replay logs of observed beacons that are generated by the localization system. This enables us to perform repeatable experiments using trace data for measurement as well as debugging purposes.

UDP Beacon Handling: UDP beacons are processed by a pair of modules: UDPDiscovery and UDPBeacon. This pair of discovery and advertising modules handles the broadcasting of and monitoring for in-band UDP beacons that describe spheres available in the local network. These beacons are transmitted as UDP broadcast packets. The beacon provides sphere identification information about the type of sphere, and also relevant application information. In the current Panoply implementation, these beacons are most commonly used to form dynamic interest-based spheres.

Multicast Beacon Handling: Multicast beacons are processed by the MulticastDiscovery and MulticastBeacon modules. This pair of modules operates in much the same way as their UDP counterparts; however, instead of UDP beacons, these components utilize a beacon transmitted via a well-known multicast group. This allows discovery in multicast-enabled networks with a narrowly defined broadcast network, such as many 802.11 portals.

Discovering Locations and Location-based Device Communities in Panoply

Localization and the detection location-based device communities are important services that Panoply provides. Techniques for inferring location context have improved dramatically as the state-of-the-art has moved from ultrasonic and infrared to GPS, RFID, 802.11 and UWB technologies. To isolate the principal Panoply functionality from ever-changing localization technology, Panoply uses *semantic maps.*

A semantic map translates from a low-level observable representation reported by hardware, with little or no meaning to the average human or application developer, to a meaningful high-level abstract notion, or *semantic tag*. This mapping builds on the dichotomy of place vs. space explored by Harrison and Dourish in [Harrison1996]. For example, we could map physical coordinates, such as GPS latitude and longitude, to a symbolic notation such as "**Kevin's house**." We could also map certain observed characteristics, e.g., a pair of access points with signal strengths S_1 and S_2 to a semantic location like "**the Westwood Starbucks.**" Semantic tags, in turn, are mapped to a specific device community or location sphere that is relevant to a local device and will provide scope and context for their interactions.

Each device sphere possesses a Localization Database component which maintains a database of maps for all applications currently running. Applications can request and receive maps from an existing sphere relation through sphere events. The Semantic Location Mapper component running in each device sphere continuously monitors the surrounding environment (via 802.11 discovery events from the WiFiDiscovery module) and compares its observations with its set of active maps. The frequency of this operation is directly dependent on the frequency of input events. The location mapper generates a location event containing either the set of matching semantic tags, stamped with the appropriate database identifier, or a "void" reading, indicating that the location cannot be determined. These events may be consumed by applications or other Panoply components.

In the current implementation, our localization scheme is based on a combination of 802.11 scene analysis [Bahl2000] and attenuation monitoring; the details of our approach are in [Schnaider06]. However, other localization schemes, e.g., [Boriello2005, Capkun2001] could be easily layered on top of our semantic localization facility and leveraged by our framework.

Dynamic Interest Sphere Formation and Leadership Management

As devices within a community interact, it may be desirable to form dynamic sub-groups to organize and coordinate local factions for purposes of collaboration and scaling. When these interest- or task-based groups form, some sphere must take on a leadership role. When multiple different devices may simultaneously express interest in joining such an interest-based sphere, the formation of the sphere needs to occur in as orderly and organized a fashion as possible. Simultaneous discovery and automatic connections can result in simultaneous bidirectional connection and negotiation attempts; in a crowded network, this can cause excessive overhead. While Panoply does attempt to detect multiple bi-directional join attempts and will prevent livelocks, sphere formation algorithms should be designed to avoid this type of behavior wherever possible. We have designed an algorithm to handle interest-based sphere formation and leadership management.

The interest group formation algorithm is designed to deterministically arrive at a stable arrangement of spheres, as well as reduce the number of connection attempts and outstanding connections within a subnet. When a sphere determines it is interested in joining an interest-based sphere (IBS), it begins to listens for a beacon from a relevant

IBS. If it hears a beacon, it connects to the advertised IBS. If the sphere fails to hear a beacon, it begins to issue its own beacon, and also continues to monitor for others' beacons.

Once the sphere is in the discovery and advertising phase, two spheres may mutually discover each other near-simultaneously. To prevent redundant connection attempts, the sphere with the lexicographically greater IP address attempts to connect to the other peer. If there are only two connected peers, there is little need for any further management—they communicate with each other, sharing events related to their common interest. However, once three or more peers are connected, this process would create a fully connected mesh, which, while fault tolerant, may result in excessive overhead.

When a sphere has two or more connected peers in an IBS, it enters an election phase. The distributed election is determined by a stability metric, intended to select the most stable leader. This process is intended to minimize thrashing when there are changes in leadership. Connected members advertise their stability metric; currently the member with the highest metric wins, similar in essence to Garcia-Molina's Bully Algorithm [Garcia-Molina1982].

Panoply currently uses the difference between each node's current time and the time its network address was last bound as a rough stability metric. This forces the group to select as leader the node whose network address changed most distantly in the past. In groups that are partially composed of infrastructure-based spheres, the infrastructure node typically wins. This outcome is ideal, since infrastructure-based nodes are typically extremely stable. Different IBSs might choose to use different stability metrics—some might want to pick a node with a median arrival time, others might pick the one with the most power, still others might choose the node with highest degree of connectivity, and so on. The leadership selection metric can be easily extended to use any arbitrary metric that the application utilizing the IBS desires; however all participants must use a common metric for a given IBS.

Once a leader is selected, all members of the IBS connect to the leader, if they are not already connected. Once an IBS member is connected to the leader, it disconnects from other IBS members if the connection is only present for the purpose of interacting in the IBS. This is intended to reduce the amount of overhead each IBS member experiences. If desired, IBS members may maintain one or more extraneous connections to other IBS members. These connections may remain quiescent, except for heartbeats. These extra connections help with stability and quick sphere reestablishment in the event the sphere leader goes away.

There is a security issue inherent in the leader selection process. Since the individual sphere advertises its own leadership metric, there is a potential for deceit; a sphere can lie about its leadership metric in order to steal leadership or reject it. The latter is less of a problem, since a sphere is not required to take on the leadership role, and rejecting the role would not be significantly harmful to the operation of the IBS. However, stealing leadership of an IBS would allow a malicious sphere to interfere with event flow in the IBS. To mitigate this, a secure protocol could be used that forces participants to commit cryptographically to their leadership metric without seeing the others' metrics. Then the group members each reveal their metrics, and the member with the median (or other agreed-upon value) is elected leader. This would prevent a sphere from setting their metric excessively high and automatically winning the leadership role. Despite this, however, a sufficiently large group of malicious spheres could still steal an election by drowning out the non-malicious nodes, forcing one of their number to be chosen. If this is a serious concern for an application, the application can limit IBS formation to known or otherwise trusted spheres either through scoping advertisements to trusted social spheres, or by specifying appropriate negotiation and connection policy.

Connection Management Service

Connections between related spheres are maintained through a SSL-secured TCP connection. Since peers will not necessarily have X.509 certificates signed by a known certificate authority, Panoply uses self-signed certificates as input to the SSL engine. Once in the negotiation phase of the connection, the peers must determine authenticity through other means, such as the verification of possession of acceptable vouchers, or by verifying the peer has completed a side-band enrollment process.

Timely detection of sphere connection failure, e.g., due to sphere mobility, is difficult due to the behavior of TCP. In order to provide timely feedback regarding disconnections, Panoply utilizes an adaptive keepalive mechanism. At a configurable time increment, related spheres exchange heartbeats, and measure the inter-heartbeat arrival time. If the sphere does not receive a heartbeat from its peer within a given time window, the connection is determined to be broken. This timeout value is adaptive; the actual value is based on the recent history of heartbeat inter-arrival times—this addresses high latency connections, where a simple fixed timeout value would improperly signal a broken connections.

When a connection is broken, the ex-child sphere may choose to attempt to re-establish the connection. This option is set by a flag in the original Join request sent by the application or Panoply system component.

Time Discontinuity Management

Many components in the Panoply active middleware have a time-based component. This includes localization, 802.11b discovery, sphere heartbeats, and others. Our primary device target is mobile devices, which may enter sleep states with little or no warning. When a device enters and exits a sleep state, Panoply components may have inconsistencies; for example, a localization component that depends on a time-windowed analysis may possess partially stale data and partially fresh data. To solve this problem, Panoply possesses a time discontinuity management subsystem, called the *TimeWarpDetector*. This component monitors for time discontinuities and triggers a system event when a discontinuity is detected. The monitoring is done by a thread sleeping for a fixed amount of time, and checking the system clock before the thread is put to sleep and after it awakes. If there is a discrepancy that cannot be accounted for, a *TimeWarp* event is submitted into the sphere. Panoply components which are dependent on a consistent view of time will have subscribed for these events, and will react accordingly. For example, our localization facility flushes its observed beacon measurements window; this prevents old data from interfering with localization and speeds the re-localization process. Applications may also make use of this facility, if they so desire.

3.1.3.5 Policy Management

Device communities typically consist of heterogeneous devices running a variety of applications; devices encounter many different situations due to the large variety of application-driven interactions. The sphere provides a natural scoping boundary for security and access policy, and one of the principal functions of this boundary is to constrain membership. When groups interact, policies must be resolved across their boundaries, potentially affecting application behavior. Many different policy constraints are possible, and they may change in response to group context changes, necessitating dynamic policy management.

Panoply provides policy management, including contextual interpretation and resolution of policies, by a per-sphere *policy manager* module. This policy manager and the associated policy language are the dissertation research of V. Ramakrishna; details and an initial evaluation are available in [Ramakrishna07]. What follows is a brief, high-level description of the impact of his research on Panoply.

We cannot rely on user intervention, nor can application designers plan for every eventuality. General policy management allows designers to focus on application specifics without having to deal with potential policy conflicts. Designers can frame application-specific policy, or intent, in terms of groups, high-level resources and actions, without specifying enforcement mechanisms. This has the benefit of widening the spectrum of applications we can support and the situations in which they can be deployed.

Panoply applications may require communication of information or content among entities who may or may not have a prior relationship, but who nonetheless share some common traits or goals. The policy manager mediates sphere interactions and regulates group membership through a *negotiation* protocol consisting of requests, offers, and policy statements. This protocol results in a safe and mutually acceptable relationship, possibly involving resource sharing, through decentralized policy resolution. Decisions at every step are guided by local policies, risks, and incentives. Within a sphere, the policy manager monitors system state continuously and triggers suitable actions based on events. It also mediates event flow, monitoring and filtering events destined for applications. These tasks are carried out using only the information specified in the system policy rules and knowledge gained by event monitoring. Our policy language is based on Prolog, policies are written in restricted Prolog syntax, and resolution algorithms are based on well-known AI algorithms. Sphere state and goals are described as Prolog facts and rules. Attestations and capabilities are passed using the voucher data structure, as detailed above.

Panoply leverages a number of policy-based services that are provided by an integrated policy engine component, including membership mediation, policy-based event-condition triggers, and dynamic event mediation and re-negotiation.

3.2 The Panoply Architecture

This section will describe the components of the Panoply architecture and how they relate to the larger Panoply infrastructure. Where the last section discussed Panoply's functionality in terms of the specific pieces of functionality it offered, this section breaks down the Panoply architecture and discusses each of the individual architectural components. Panoply system components are organized into six major systems, as depicted in Figure 9.

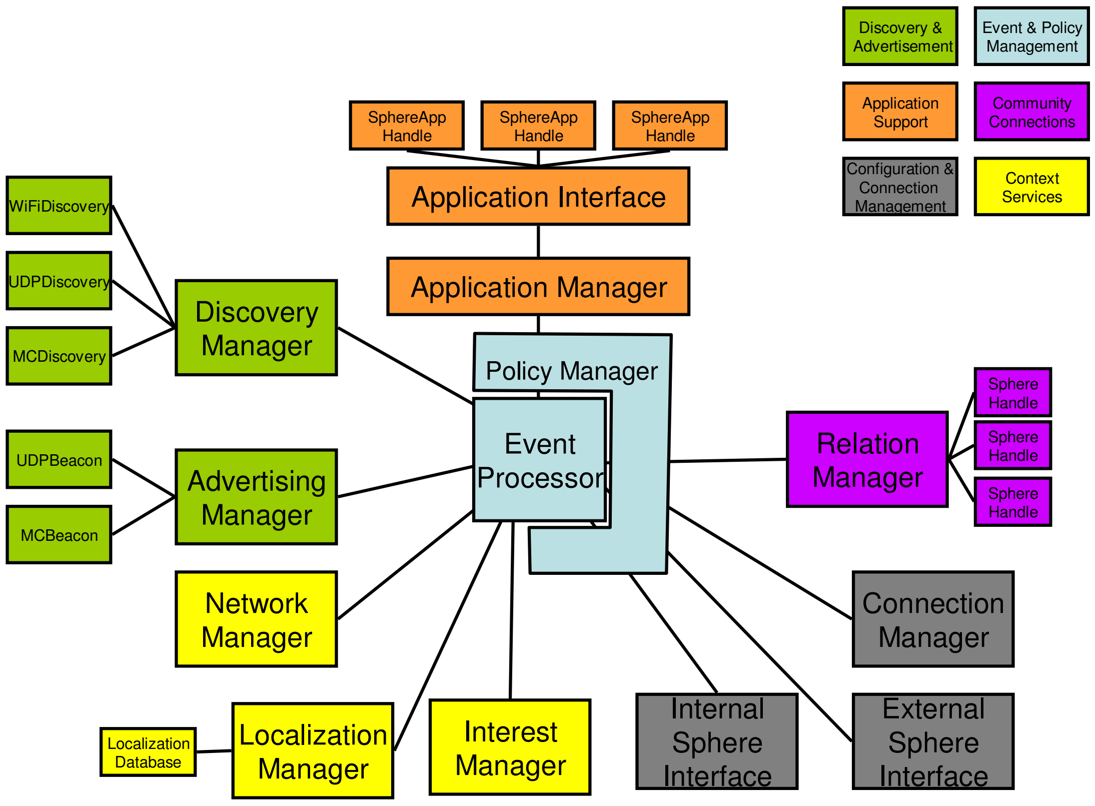


Figure 9. High-level Panoply architecture

3.2.1 Events and Policy

The event framework is the core of each Panoply sphere. As discussed in Section 3.1.3.1, all Panoply components communicate with one another using events. Panoply supports two types of event forwarding—interest-based forwarding and source-route-based forwarding. Policy management is integrated with event passing; events passed to and from applications and remote spheres are mediated by policy. Additionally, the sphere connection process is dependent upon the successful completion of a policy-based negotiation process between the supplicant sphere and the target sphere.

3.2.1.1 EventReceiver

The EventReceiver class is extended by every Panoply component that wishes to receive events. EventReceiver implementers must implement three methods:

`int[] getEventInterest():`

This method must return an integer array encoding the set of system event types that the implementer wishes to receive. The integer values for the event types are defined in the Event class.

`String[] getUserInterest():`

This method returns the set of user-defined event types the implementing class wishes to receive. These are encoded as strings, e.g., "MyEventType." These are hashed internally for efficient processing.

`void queueEvent(Event e):`

Every event-receiving component must implement an event queue and the corresponding queueEvent method. This method must be non-blocking, and ideally just pushes the event into a data structure and returns; this prevents the event processor from blocking during the event dissemination process. The receiving components must service their queues with another thread.

3.2.1.2 Event

The Event class is the base unit of communication in Panoply. This class can be subclassed to create custom event objects, but every message that is passed around is ultimately encapsulated inside an event. Events are defined primarily by the following fields:

`EventType type:` The principal event type of the event. This is used to route the event to interested event receivers.

`EventSubType subtype:` The secondary type of the event. This is used by receiving components to direct the event to the appropriate handler without requiring heavyweight **instanceof** checks.

`String userType:` The user-defined event type of the event. This is optional, and if it is not set, the default value of *null* will cause this field to be ignored.

`Serializable Attachment:` Events can possess an optional attachment. The attachment can be any object that implements the Java *Serializable* interface, allowing the event (and its attachment) to pass between spheres and to and from applications. The attachment is used to carry data, encapsulate other events, etc.

`EventScope scope:` The scope specifies bounds on the propagation of the event. Absent a specified scope, the event will propagate indefinitely until there are no more interested event receivers. Panoply supports a variety of event scopes, as has been discussed previously. An obvious extension on the existing scopes would be a composition mechanism that allows event scopes to be combined, enabling more flexibility in event distribution.

Events include other valuable information that is used by Panoply, including the *eventProducer,* which identifies the sphere or application that created the event, and the *visitedSpheres* path, which encodes the ordered set of spheres that the given event object has traversed.

The DirectedEvent

Directed events are a special type of Panoply event that specify a forwarding path. The directed event is wrapped around the event that the Panoply application or component desires to send to the specified recipient. The event creator must specify the appropriate path of spheres along which the directed event must be routed in the form of an array of sphere identifiers, or *SphereIDs*, unique identifiers possessed by every sphere. Directed events are then passed via the event processor to the relation manager, which routes them to the external sphere, as described later in Section 3.2.6.2.

3.2.1.3 Event Processor

The Event Processor manages event subscriptions and event dissemination within a Panoply sphere and between related spheres. The Event Processor maintains an event subscription table that maps registered components, of base type *EventReceiver,* to the event types they are interested in receiving. Event registration is handled through *REGISTER* events which are processed internally by the Event Processor to build the registry table entries. Incoming events are analyzed based on their event type, and passed through the subscription table to identify interested receivers. The resulting set of event receivers is filtered based on the event's scope; compatible event receivers are passed a copy of the event object via the queueEvent() method in the *EventReceiver* interface. An object copy is used to protect the receiving components within the same Java Virtual Machine (JVM) from the actions of other receivers that could change the state of the event.

3.2.1.4 Policy Manager

All direct interaction with the Policy Manager occurs via *POLICY* events. The Policy Manager manages the policy database, handles the negotiation phase of sphere joins, and mediates event passing to and from applications and other spheres. Negotiation and event passing is policy-mediated in order to ensure compliance with local sphere policy. Panoply policy management will be briefly described, as it is fundamental to the operation of a sphere, but it is important to reemphasize that the policy manager and the associated policy language are not a part of this dissertation, but are the focus of a related dissertation [Ramakrishna2007].

Local Sphere Policy

Our policy framework assumes that the interacting spheres possess policies that describe their state and constraints. Panoply policies are sets of rules that describe the behavioral constraints and goals of the sphere. Examples include policies for resource management, security and access control, and content adaptation.

The Panoply policy language is built on Prolog. The structure of predicates, variables and other terms in Prolog allows us to specify categories and instances of entities, objects and contextual parameters in policy rules. The general semantic nature of a logic-based policy language enables the specification of high-level policies that build upon low-level policies. The Policy Manager resolves Prolog queries natively using SWI-Prolog [SWIPL] and SWI-Prolog's JNI interface. System state and policy rules are defined in the form of Prolog statements and clauses. Examples are included in Figure 10, with explanations in square brackets. Clauses (rules 4 and 5) indicate an *if-then* policy.

1) *fileType('song.mp3',audio).* ['song.mp3' is an audio file]
2) *relation(alice,bob).* ['alice' and 'bob' are relations]
3) *certificate('UCLA').possess(john,'UCLA').* ['UCLA' is a certificate, and is possessed by 'john']
4) *member(X) :- candidateSphere(X), teamMember(X), numChildren(N), maxChildren(M), N<M.* [X is a member if it is a candidate sphere, and a team member, and if the number of current children is less than the maximum]
5) *access(S,V) :- candidateSphere(S), teamMember(S), voucher(location,V).* [sphere S can be granted access to voucher V if S is a candidate sphere, and is a team member, and if V is a 'location' voucher]

Figure 10. Example Panoply Policy Statements

Policy Negotiation

Negotiation is a policy-guided operation by which spheres request and grant services from and to each other. Each participant's local policies are private and unknown to the other, and there is always a possibility of conflicting policies. Policies are kept private to protect both negotiating spheres. Each negotiating entity starts with certain requirements and goals; negotiation guides entities to agreement or a mutual *compromise* through the use of meta-policies, heuristics, and logical reasoning. The negotiation process is a decentralized process of policy resolution and conflict management, where each negotiating entity possesses only partial knowledge of the other's policy, state and goals. The negotiation protocol is bi-directional between the negotiating spheres, as both entities may possess resources or privileges that the other may desire. For example, a patron's PDA and a coffee shop network could derive mutual benefit from interaction; the former obtains network access, while the latter could expand its customer base through incentives that include network access.

Negotiation can occur at any stage of communication between two spheres, although it always occurs in the *join* and *accept* of the supplicant and leader sphere state machines, respectively. The end result of this negotiation during the join process is either an APPROVE or DENY event that approves or rejects the supplicant sphere's admission into the target sphere. Other forms of negotiation can include dynamic negotiation for a new access right, such as the privilege to send or receive a certain event type, and also re-negotiation, when sphere context changes and the peers need to re assess access rights. Additionally, vouchers granting privileges may also be exchanged. These vouchers can later be presented as secure attestations of privilege, such as during event filter negotiation; in this role, vouchers act as *distributed capabilities* [Tanenbaum1986].

Event Mediation

In its other primary role, the Policy Manager acts as an intermediary for events destined for applications or remote spheres. Events are checked by the policy manager against the current policy rules for consistency with the rule set. Events that are found to be inconsistent with policy are suppressed, or alternately, the event sender is asked to show authorization to send the specified event through negotiation or the presentation of an authorizing voucher. This functionality allows spheres to interact in a late-binding fashion, only negotiating for a certain privilege when the privilege is in fact needed; this is in accordance with the *least-privilege* security paradigm. If the sphere cannot successfully negotiate to send its event, the submitted event is suppressed.

3.2.2 Application Support

The second major component of the Panoply architecture is the application support system. Applications leverage the Panoply middleware to interact with their environments and social contexts, and other devices within those contexts. Panoply applications connect into their local sphere via a loopback network connection, and communicate with Panoply by exchanging events with their local sphere.

3.2.2.1 Application Manager

The Panoply SphereAppManager is the application manager component. This component manages and controls applications attached to the local sphere. The SphereAppManager abstracts away the management of the loopback application interface with the *SphereAppInterface*, and provides an EventReceiver-based abstraction, the *SphereAppChannel*, that allow applications to act as event receivers. Additionally, the SphereAppManager manages several other important aspects of Panoply applications.

Internal Application Loading: The ApplicationLoader component launches applications in response to ApplicationLaunchEvents of type *COMPONENT*. These events specify the application class to load, as well as any parameters to pass to the constructor. This allows Panoply components to launch applications. For example, this feature is used to start applications specified inside configuration vouchers when a relevant environment is detected.

Application Instance Control: This feature allows applications to specify limits on the number of instances of a given application that may be instantiated simultaneously.

Applications may be automatically launched based on user preferences and community profiles. Instance control prevents redundant application instantiation. The application manager enforces instance control, and will reject any application connection that exceeds the specified instance limit for that application.

Ad hoc Rescue Mode: Device spheres may enter *ad hoc rescue mode* to assist other spheres they have detected in need of ad hoc service. However, entering ad hoc rescue mode at the wrong time may interfere with an operating application. The application manager supports a *canEnterRescueMode()* method that queries the connected applications to determine if they are willing to allow the sphere to enter rescue mode. The ad hoc rescue functionality will be discussed further in Section 3.2.4.

3.2.2.2 Panoply Applications

Applications that wish to interact with Panoply are required to extend the *SphereApp* base class, which itself extends the EventReceiver class. SphereApp handles setting up the network connection to the local sphere and abstracts the event communication between the sphere and the application. The application sends events by calling *sendMyEvent(Event evt)*; this method calls the underlying *SphereAppHandle*'s queueEvent() method to queue the event to the local sphere. Events are properly relabeled, if necessary, associating the event with originating application. This enables

```
public class SampleApp extends SphereApp {

//Default constructor
    public EventTest() {
        super();                //SphereApp constructor
        initialize(this);               //Connect to local sphere
        setSphereApp(this);             //Setup callback handlers
        setAppName("SampleApp");        //Set the app  name
        registerInterest();     //Submit our event interest
        }

//Register event subscription
void registerInterest() {
        EventMatcher ed = EventMatcher.createEventMatcher(
             new int[]{EventType.APPLICATION},
             new String[]{"SampleAppEventType"});

        //Inject the registration event into the sphere
        sendMyEvent (new RegistryEvent(ed));
}

//Callback to handle delivered events
//Whenever the sphere sees an event that matches our interest,
//it will pass it to us through this callback


@Override
    public void queueEvent(Event evt)  {
   .
   .
   .
}
```

Figure 11. A simple Panoply application

spheres or applications receiving the event to respond, when necessary, to the actual event sender.

Applications receive events by implementing their own *queueEvent(Event evt)*, overriding the implementation in the EventReceiver base class.

The sample application depicted in Figure 11 is the simplest Panoply application possible. It demonstrates the three components necessary for a Panoply application to initialize, connect to the Panoply middleware and register to receive events.

The parameter-less constructor is the default constructor that can be dynamically loaded and called by Panoply if this SampleApp were launched from inside a Panoply sphere via the Panoply ComponentLauncher. Within the constructor, the application initializes the SphereApp parent class, which contacts the local device sphere via the loopback interface and registers as an application. This sets up an appropriate SphereAppChannel handle within the event processor. This allows our SampleApp to register its interest via a RegistryEvent. When events that match the application's subscription pass through the EventProcessor, a copy of the event is handed off to the SphereAppChannel which serializes it out to the application's local sphere handle, which in turn calls the registered application's *queueEvent* callback.

3.2.3 Discovery and Advertisement

The third major component of the Panoply architecture is the discovery and advertisement subsystem. This component is critical to Panoply's ability to form device communities. Panoply offers an extensible discovery and advertisement model, allowing developers to add new discovery or advertisement modules that may interact with internal Panoply components and also Panoply applications.

3.2.3.1 Discovery Manager & Advertising Manager

The Discovery Manager organizes all of the discovery plug-ins, and the Advertisement Manager handles the corresponding advertising plug-ins. The managers determine the set of relevant discovery and advertisement plug-ins from the Panoply properties, accessed via their user-specified sphere preferences. These plug-ins are dynamically loaded, instantiated, and connected to the sphere. The managers support starting and stopping their respective sets of plug-ins, and allow applications to submit new advertisement data to the running plug-ins, dynamically changing the advertisement content.

The Discovery Manager has a secondary duty, which is to monitor the network connectivity of a specified interface and generate connectivity updates when the connectivity status changes; e.g., are we associated with a wireless network, do we have an IP address, do we have a gateway, is the gateway reachable, etc? This information is used to assist with determining the need for reconfiguring the network interface.

The plug-ins will be briefly described again here for completeness. For more detail regarding the functionality of the discovery and advertising plug-ins, consult Section 3.1.3.4.

3.2.3.2 Discovery Plug-Ins

WiFiDiscovery: The WiFiDiscovery plug-in is used to detect 802.11 beacons in the surrounding environment. Beacon measurements are taken at configurable intervals (e.g., 2 seconds) and are injected into the local sphere as discovery events. These events are typically of interest to the Network Manager and Location Manager components which use them to discover relevant communication spheres, determine location, and identify relevant location spheres.

WiFiTraceDiscovery: The WiFiTraceDiscovery plug-in processes trace files containing logged 802.11-beacon traces generated by the localization subsystem. These are handled in an identical manner to real data from the WiFiDiscovery component, and can be used for repetitive testing as needed for simulation or debugging.

UDPDiscovery: The UDPDiscovery component monitors for UDP broadcast packets that contain sphere identifiers and application data. These are turned into discovery events that are typically consumed by the Interest Manager as well as interested applications. These are used to detect peers with common interests or applications, with which the sphere may form an interest-based sphere for collaboration purposes.

MulticastDiscovery: The MulticastDiscovery component functions in a similar manner to the UDPDiscovery component, except instead of using UDP broadcast packets, this component subscribes to a well-known multicast group (225.1.1.1:5600) and listens for sphere announcements. These are processed in a similar manner to the UDP discovery beacons by the Interest Manager as well as interested applications.

3.2.3.3 Advertising Plug-Ins

UDPBeacon: The UDPBeacon component generates the UDP broadcast beacons that advertise the sphere's presence and participation in a specified application or interest. Applications can generate *AdvertisingControl* events that specify the interest or application that the sphere will advertise, as well as a directive to start or stop the specified advertisement. The UDPBeacon component encodes the advertisement data and the relevant *SphereID* into an advertisement packet and transmits it into the network.

MulticastBeacon: The MulticastBeacon is identical to the UDPBeacon in functionality, except that instead of transmitting a UDP broadcast beacon, a multicast message is used instead to announce the sphere's interests and presence.

3.2.4 Context Services

The fourth component of the Panoply architecture is a suite of context services. Panoply provides three primary context services to assist devices and their applications with creating and joining relevant device communities in order to acquire connectivity and network services, participate in communal applications, and coordinate application activities. These context services receive input from the discovery services just described, and interact with the connection and communication facilities that will be described in Section 3.2.5. The three primary subsystems are the Network Manager, the Location Manager, and the Interest Manager.

3.2.4.1 Network Manager

The Network Manager is responsible for matching observed external network conditions against a set of vouchers that describe viable device community configurations for a set of known networks, and issuing requests to alter the network configuration and join specified spheres when applicable. The manager component maintains a database of vouchers that describe communication sphere profiles, including network configuration, valid time, the address of any associated device community, and relevant applications.

The Network Manager receives external observations in the form of 802.11 beacon measurements and network connectivity updates. If the local sphere does not have network connectivity, or does not have a local communication sphere, the observations are compared against the stored voucher database. If the observation contains a

<MAC,SSID> tuple that matches a database entry and the current time is within the valid time parameter specified by the entry, the entry is considered a match.

If there is a single match, the Network Manager activates the matching entry by generating a sphere *JoinRequest* with the appropriate configuration information (network configuration, communication sphere details, and applicable applications) and submitting it into the sphere. If there are multiple matches, the user is prompted to select between them, and the selected entry is activated. The selection process could be automated through the use of a preference-based heuristic. If there are no matches, the Network Manager continues to process observations.

A secondary function of the Network Manager is to monitor for rendezvous rescue requests. This is a relatively recent feature of Panoply, in which a device sphere in need of assistance can configure their wireless interface to a sphere rendezvous setting—an ad hoc 802.11 cell with a well-known SSID. Spheres that observe this identifier may, if they are able, configure themselves to the ad hoc cell, assign themselves a IPv4 link-local IP address and form an ad hoc sphere. Within this sphere, devices may collaborate, exchange location- and application-specific data. The Locative Media Application, discussed in Chapter 6, details one application of this feature.

The Network Manager is in charge of monitoring for these rescue requests, and determining if the device sphere is able to make a rescue attempt. It does this by querying application callbacks to determine if the current Panoply applications are willing to allow the rescue attempt to be made. This allows applications that are waiting for data or in the middle of an operation to delay the rescue proceedings. In the future, these callbacks will also let Panoply query the Panoply applications to determine if there is any actual value in assisting with the rescue.

3.2.4.2 Location Manager

The Location Manager is comprised of a collection of localization and location management facilities within Panoply. The principal components include the *SemanticLocationMapper* and *SemanticLocationScanner*. Additionally, there is an external application, the *SemanticLocationScannerApp* that is used to interface with the SemanticLocationScanner.

SemanticLocationMapper: The SemanticLocationMapper receives incoming discovery events representing observations of 802.11 beacons. The observations are processed against one or more localization databases using techniques discussed previously in section 3.1.3.4; if the localization algorithm matches the observations against the database(s), the resulting <*semantic tags, database ID, community info*> tuple is published in a LOCATION event to interested components. A sphere *JoinRequest* event for detected location-based spheres may optionally be automatically generated. Location events are separated into updates and refresh events, depending on whether there has been a detected change in location state.

Additionally, the SemanticLocationMapper processes localization configuration events. This includes injecting new localization databases, as well as reloading the database. When a device sphere enters a new communication-based community, the community may provide localization maps, allowing the sphere to dynamically extend its range of operations to the appropriate location spheres.

SemanticLocationScanner: The SemanticLocationScanner component, used in conjunction with its application interface, the SemanticLocationScannerApp, creates localization maps by logging incoming observations into a tagged data structure and then writing them out into the appropriate database files.

3.2.4.3 Interest Manager

The Interest Manager manages membership in interest-based communities, connecting and disconnecting from interest spheres based on application needs. The Interest Manager receives discovery events from UDPDiscovery and MulticastDiscovery components that describe other spheres and their interests and applications. Additionally, the Interest Manager submits advertising requests to the Advertising Manager to advertise local application interests. The basic interest sphere management algorithm is described previously in Section 3.1.3.4 under "Dynamic Interest Sphere Formation and Leadership Management."

3.2.5 Configuration and Connection Management

The next Panoply subsystem that we will discuss is the Configuration and Connection management system. Sphere interactions are separated into two interfaces—the External Sphere Interface and the Internal Sphere Interface. This design was chosen primarily for two reasons—flexibility and security. We will further discuss this design choice further below. A third component, the Connection Manager, handles join requests and manages network interface configuration. Finally, the Voucher Manager, a security subsystem that generates and manages the local set of vouchers, also resides in this space.

3.2.5.1 External Sphere Interface

The External Sphere Interface (ESI) is a sphere's central point of contact for external spheres. A supplicant sphere contacts another sphere via that sphere's ESI, by default at port 1066. This connection is TLS-based and uses RSA public-private keypairs encoded in self-signed X.509 certificates to bootstrap the TLS session. Once key negotiation is complete, a secure *ESISession* begins, in which the external sphere's candidacy is evaluated based on policy. If the sphere is allowed to become a candidate in the sphere, it is provided with contact information for the Internal Sphere Interface. This allows the sphere several options for the location of its interfaces; in theory, they can potentially lie in different virtual machines or even on different hardware. This provides the sphere with substantial flexibility in its structure, and can be used to create a security boundary around the internal sphere. In practice, our prototype does not currently support separating the interfaces onto different machines; however Panoply is designed to make it a natural and simple extension.

3.2.5.2 Internal Sphere Interface

The Internal Sphere Interface (ISI) is the network interface that sphere members use to connect into the sphere and interact with the sphere. After being transferred from the ESI, the supplicant sphere interacts with an *ISISession* (again TLS protected) that performs basic negotiation, determining whether the supplicant sphere is an acceptable sphere member based on local policy. This negotiation process might include:

- The supplicant provides vouchers that indicate the right or permission to join the sphere, signed by an authoritative source.
- The supplicant provides authorizing vouchers indicating that it has participated in and completed a local enrollment process.
- The supplicant demonstrates that it is running specific versions of software or system services.
- The supplicant promises to behave in a specified manner; e.g., it will not share copyrighted content; it will not play audio above a specified volume; it will not access content above a PG rating, etc.

Additionally, the supplicant may also make requests of similar character from the target sphere, to which the target sphere may or may not agree.

After a successful negotiation, the ISI wraps up the associate sockets into a *SphereHandle* and hands the handle to the *Relation Manager,* which will manage the connection.

3.2.5.3 Connection Manager

The Connection Manager, also known internally as the *Doorman*, is responsible for initiating new connections to external spheres and handling basic network configuration tasks. The Connection Manager subscribes for a number of different types of events, but it is primarily interested in DOORMAN events. These events include *JoinRequest*, *JoinSuccess, JoinFailure,* and *JoinError* events. DOORMAN events register requests to connect to new spheres or networks, and report the results of outstanding connection attempts.

The *JoinRequest* event is used by Panoply components to initiate a connection or configuration change. The event may contain network configuration information, sphere contact information, and pertinent applications to be executed in the context of the new configuration. The Connection Manager extracts the relevant data from the event and initiates new connections and configuration changes as necessary to satisfy the event parameters. If the device already possesses an active, working networking configuration, network configuration changes are ignored unless a *force* flag is set in the event.

3.2.5.4 Voucher Manager

The last component of the configuration and connection subsystem is the Voucher Manager. The Voucher Manager (VM) performs a number of important duties, including managing the database of vouchers, as well as voucher creation and transmission of vouchers to connected sphere relations.

Voucher Generation: The VM generates cryptographic vouchers in response to requests from the Policy Manager, the enrollment application, or configuration tools. These vouchers may encode attestations of sphere relationships or sphere properties, or permission to join a specified sphere; they may also encode network and application configurations to be activated upon observing certain triggers.

Voucher Database Management: The VM also maintains the local voucher database that contains configuration vouchers, as well as attestation vouchers. The VM provides access methods that allow the voucher database to be queried. Additionally, the VM handles automatic removal of vouchers that have expired.

Voucher Escrow and Delivery: In addition to storing vouchers for local use, the VM stores vouchers that are intended for use on other spheres. When a sphere connection is made, the VM receives a notification. It checks the identity of the relation—if the VM possesses any outstanding vouchers that are intended for the remote sphere, those vouchers are transferred to the remote sphere.

3.2.6 Community Connections

The next Panoply subsystem handles existing sphere connections. Once a network connection to a sphere has been established, it must be managed. These network connections are abstracted into SphereHandles, which in turn are managed by the Relation Manager.

3.2.6.1 SphereHandle

The SphereHandle abstracts the network connection to the remote sphere, and provides an *EventReceiver* interface to the remote sphere. Each connected sphere is represented by an instance of a SphereHandle. This allows SphereHandles to register event interest with the EventProcessor and receive events via their respective *queueEvent()* methods. Upon establishing their SphereHandle to their peer, the parent and child spheres exchange event subscription information with one another through the SphereHandle.

Additionally, the SphereHandle monitors the liveness of the network connection to the remote sphere using the adaptive heartbeat-based keepalive described earlier in Section 3.1.3.4.

3.2.6.2 Relation Manager

Once the sphere connection has been turned into a SphereHandle, it is handed off to the Relation Manager. The Relation Manager (RM) maintains a set of data structures representing parents, children, and siblings. These are used for building the membership tables necessary to answer *MembershipQueryEvents* that may be generated by applications or other Panoply components. Additionally, the RM monitors the SphereHandles for their liveness state. When a SphereHandle is disconnected, due to either being killed by the *HeartbeatMonitor* or due to an explicit disconnection event, the RM cleans up the handle's state, including outstanding events and subscription information.

The Relation Manager's other main duty is to handle *DirectedEvents*, as described in Section 3.2.1.2. When a DirectedEvent hits a sphere, it is immediately delivered to the RM who determines if a) the event is destined for the local sphere, or b) if the next hop in the path is a connected sphere. If neither of these is true, the DirectedEvent is discarded. Future enhancements will include delivering a failure notice to the event originator, if possible.

If the directed event is addressed to the local sphere, the encapsulated event is extracted and submitted back to the EventProcessor for handling. Otherwise, if the local sphere is connected to the next sphere on the directed event's path, the event is passed to that sphere's SphereHandle.

Chapter 4

Panoply as an Application Platform for Ubiquitous Computing

In its different incarnations, the Panoply middleware has been in use in our laboratory for approximately three years, and has been used by more than ten different developers, including both graduate students and undergraduate students. This section discusses Panoply's role in the development of various pervasive computing applications. Section 4.1 presents the system requirements for Panoply, as well as key performance measurements for core Panoply operations, including communication and configuration operations, network discovery, and application startup. Additionally, this section discusses our experiences with porting Panoply to a variety of different systems.

We have developed multiple applications and services to illustrate the basic principles of Spheres of Influence and Panoply, and have used these in our lab and around the UCLA campus. Three of these, QED, the Locative Media Applications, and the Smart Party, will be discussed in depth, each in their own chapter (Chapters 5 through 7 of this dissertation). The remaining applications will be discussed in Section 4.2. We will also explore how these applications leverage Panoply features, and how Panoply helped simplify application development.

4.1 System Requirements and Panoply Performance

4.1.1 Panoply Requirements

Panoply imposes modest demands upon target devices. However, Panoply currently operates best on personal-computer-class machines. It does not yet perform well on more restricted devices, although PDAs and mobile phones are swiftly developing into suitable platforms.

Memory Requirements

We have been able to reduce a running Panoply sphere to a minimal memory footprint of 26 MB—this includes the overhead of the Sun Java Virtual Machine (JVM). Using a special-purpose JVM, it would certainly be possible to reduce this further. The typical footprint of a device sphere that is actively localizing using our existing localization plug-in with an in-memory localization map is 78 MB, including the overhead of the JVM. Our current localization plug-in is not particularly optimized for memory use; its memory usage could likely also be reduced further.

CPU Requirements

Panoply does not impose any particular CPU requirement. In practice, we have seen acceptable performance with a minimal configuration of a 466 MHz Intel Celeron CPU. More limited processors present challenges to fast execution of interpreted Java bytecode. CPU-specific Java extensions such as the Jazelle Execution Environment Accelerator [Jazelle] for the ARM9, when paired with a compatible JVM, may allow Panoply to perform acceptably on these platforms.

Storage Requirements

In addition to whatever storage requirement is necessary for the platform-specific Java Virtual Machine (e.g., 133 MB for Sun Java 6 on Linux x86), a minimal Panoply installation requires approximately seven megabytes of storage. This includes the Panoply binaries and necessary supporting software libraries. Additionally, localization maps require some storage. A sample map that describes the Boelter Hall suite 3564 environment requires ~330 KB of storage.

Networking Requirements

Panoply applications connect to their local device sphere through a loopback network interface bound to 127.0.0.1. A Panoply sphere requires a non-loopback network interface to connect to other spheres. Ideally, for device spheres that represent a mobile device, this network interface should be wireless. Panoply requires TCP/IP to connect spheres. Panoply also uses UDP broadcast and multicast packets for sphere discovery. Additionally, our localization plug-in requires an 802.11 wireless network interface. Specifically, the wireless interface must offer a reasonable implementation of the Linux SIOCSIWSCAN *ioctl* which is used to request a list of scanned 802.11 access points.

Software Requirements

Panoply depends on a number of libraries and other software packages.

Java Virtual Machine: Panoply is written in Java, and requires a Java 5 compatible JVM. For reasonable performance, it requires a platform-optimized JVM, preferably with a just-in-time (JIT) compiler.

Java Security Extensions: Panoply makes extensive use of Java security libraries—these may be derived from Sun's JSSE implementation, or open-source implementations such as Jessie [Jessie].

Bouncy Castle Crypto API: In order to process and generate X.509 certificates, Panoply leverages the Legion of the Bouncy Castle Crypto API [BouncyCastle] and associated implementation. A different cryptography package with similar functionality could be easily substituted.

SWI-Prolog: Panoply uses SWI-Prolog [SWIPL] and the accompanying Java Prolog (libjpl) package. Prolog is used for policy mediation of sphere connections and event flow.

Place Lab: Panoply currently uses a localization plug-in that leverages a portion of Place Lab [LaMarca2005], a localization system from the University of Washington. We utilize the spotter library component, which queries the wireless interface for beacon readings. This is a small, self-contained portion of Place Lab and could be replaced, if necessary, with a custom spotter component. Other future localization plug-ins may obviate the need for Place Lab entirely.

4.1.2 Measuring Basic Panoply Overhead

Basic Panoply performance measurements were collected using IBM ThinkPad T42 laptops. Each laptop was equipped with a 1.7 GHz Intel-based CPU and 512 MB of RAM, as well as an Intel Pro Wireless (IPW) 2200 wireless interface. All communication was performed over an 802.11 network managed by a Linksys WRT54G wireless router; network cards were configured to use 802.11b. Experiments were run under Ubuntu

Linux (Feisty Fox) using the Java 6.0 run-time environment from Sun Microsystems. Time synchronization was maintained between nodes using *ntp* [RFC1305] in a broadcast server configuration.

The basic overhead measurements can be broken down into the following categories: configuration and connection establishment, communication, and application startup.

Configuration and Connection Establishment Overhead

This category groups together the basic connectivity functions of Panoply. Our primary configuration measurement of interest is the time required to identify and activate a relevant configuration voucher. This process includes detecting that the sphere is in the presence of a relevant wireless network and selecting the appropriate voucher from a database of several hundred configurations, configuring the network interface with the appropriate settings, and generating an *ApplicationLaunch* event to launch any appropriate applications. Other related configuration measurements that are specific to the given application scenario are discussed in the Smart Party case study in Chapter 7. Connection establishment overhead refers to the time required for one sphere to connect to another and form a relationship. Sphere Join was measured with minimal policy. To measure these Panoply functions, we performed hundreds of repeated trials. The results of these trials are reported in Table 3 with 99% confidence intervals.

Panoply Phase	**Time (in ms)**
Network Voucher Selection	1372.9±85
Network Voucher Activation	334.3±2.2
Start-to-Finish Network Selection & Configuration	1707.2±85.4
Sphere Join Time	2705.4±108.5

Table 3. Time required for Panoply configuration and connection phases (measurements are reported with 99% confidence intervals)

The first measurement represents the time required to observe a nearby network and identify a relevant voucher that describes a valid configuration that corresponds to the current environment. The next measurement represents the time to activate the network configuration encoded in the selected voucher. From these results, we see that network configuration is dominated by the voucher selection—this is due to an interrupt-driven timer that polls the 802.11 spotter and generates WiFiDiscovery events at two-second intervals. This results in, on average, a one-second delay between arriving within proximity of a relevant 802.11 network and detecting that network. The third measurement is the combination of the first two measurements.

The fourth measurement, Sphere Join, represents the time required to establish a connection to the sphere specified in the voucher, once the device's network has been properly configured. The Sphere Join overhead is consumed primarily by the simple negotiation process that the spheres undergo in the connection process. Experiments conducted in [Ramakrishna07] with the same experimental setup indicate that the negotiation phase of the sphere connection required slightly more than two seconds—this is consistent with the result presented here, and indicate that the remainder of the Sphere Join is relatively cheap. As of this writing, efforts are ongoing to reduce the initial negotiation overhead.

Communication Overhead

We are interested in several types of general communication overhead. Within Panoply, communication happens between spheres and sphere applications. This includes communication between spheres, within spheres, between applications local to a sphere, and also communication between a local application and an application hosted by a remote sphere. To measure the communication overhead of Panoply in different communication configurations, we measured the time required to transfer an application-type event from different source and destinations points within Panoply. As the time required to send a single event was not measurable given the millisecond-level granularity of the internal Java clock, we measured the time required to transfer one thousand instances of the event. Each event was 1690 bytes in size when serialized. The different cases include: event passing within a sphere, sphere to local application, local application to sphere, local application to other local application through a sphere, sphere to remote sphere, and application to remote application (through local sphere to remote sphere connection). For this experiment, "local application" refers to a Panoply application running on the same device as the Panoply sphere, but in its own JVM external to the sphere's JVM. Each measurement was repeated several hundred times. Table 4 summarizes these results; all numbers are reported with 99% confidence intervals.

Event Path	**Time (ms)**
Within Local Sphere	13.9±3.9
Sphere to Local App	1026.1±59.5
Local App to Sphere	682.3±3.9
Local App to Local App	1634.2±22.5
Sphere to Remote Sphere	4854.5±26.5
Local App to Remote App	5823.4±80.3

Table 4. Time required to transfer 1000 events through different Panoply paths (measurements are reported with 99% confidence intervals)

From these numbers we see that, as expected, serialization and network communication represent the bulk of the time. Intra-sphere event passing is very efficient since all communication is handled interior to a single JVM, and no serialization or network exchange occurs. Event passing to applications passes through the SphereAppChannel which consults the Panoply policy manager to determine if the event is allowed to be delivered to the application. This adds a small amount of overhead in the process of delivering the events to the application.

Application Startup Overhead

Panoply applications incur some startup overhead associated with launching the application. Applications can be launched by users or by the local sphere, either within the sphere's JVM or in an external JVM. To measure the startup overhead, we conducted multiple trials involving measuring the time between launching the application and when the application was ready to receive events from the local sphere. These results are shown in Table 5.

Application Mode	**Time (in ms)**
Application co-located in Sphere JVM	7.4±1
Application external to Sphere JVM	828±3

Table 5. Time required to fully launch a Panoply application

(measurements are reported with 99% confidence intervals)

These results indicate that launching an application inside the sphere's VM is very cheap; however, since the application code is running within the Sphere's VM, there is an increased risk of compromise, either by malicious local application code, or by an external attack on the now-privileged application.

Running an application inside an external, user-level JVM adds additional startup overhead, but does offer substantial security benefits from separation of privileges. Since all communication with the local sphere is through events, executing the application within an external JVM limits the possible vectors for sphere compromise to vulnerabilities in the event channel or vulnerabilities that can be exploited through Panoply events.

4.1.3 Platform Issues

We have primarily been developing and evaluating Panoply on an IBM ThinkPad-based platform. However, a smaller and more mobile platform is better suited to the overall Panoply vision. With that in mind, we have attempted to port Panoply to several other platforms. This section details those experiences.

Nokia 770

One of our earliest attempts to port Panoply to a mobile platform was with the Nokia 770 Internet Tablet. The Nokia 770 is a palm-top computer designed to be used for casual browsing, email, and media applications. It is equipped with a built-in 802.11 interface, and a 252 MHz Texas Instruments OMAP processor which combines an ARM-based architecture with a Texas Instruments DSP. Equipped with 64 MB of RAM, this seemed like it might be a reasonable platform for Panoply, depending on how well Java performed and on the memory footprint of the JVM.

We were able to get Panoply to build in the 770 environment with some minor modifications, but ran into a major problem. Code execution was extremely slow due to lack of a just-in-time compiler (JIT) or any CPU-optimized instructions for this platform at the time. Event propagation delays were in excess of ten seconds. On-chip Java acceleration should be possible due to the ARM-based Jazelle [Jazelle] extensions within the OMAP CPU; however, existing JVM implementations do not utilize them. Improved Java support would potentially allow Panoply to operate on this device.

Certainly, successor devices, such as the Nokia N800, with its 330 MHz OMAP processor and 128MB of RAM, will make excellent Panoply platforms.

OQO and Sony uPC VX-280

In our quest to find the ideal miniaturized Panoply platform, we tested Panoply on the OQO, a full-featured miniaturized PC equipped with a Transmeta Crusoe CPU, and also on the Sony uPC, another commodity micro-PC. Panoply runs well on both platforms using the Sun JVM.

On the OQO, 802.11-based localization does not work with the internal wireless interface due to a poor implementation of the SIOCSIWSCAN ioctl in the kernel wireless driver. This ioctl is used by the spotter component to query the kernel for 802.11 AP scan results. The OQO can be used as a Panoply platform if an external, e.g., USB-based, 802.11 network interface is used for localization.

Localization on the Sony uPC operates using the internal IPW3945 wireless interface. However, unlike the IPW2200 on the IBM ThinkPads we use for development, scanning while maintaining a wireless association does not operate properly; the kernel only returns the AP entry for the associated access point. To scan properly, the interface must be put into an un-associated mode. Unfortunately, this prevents any network connectivity. As with the OQO, an external wireless network interface is necessary to use the uPC as a Panoply platform. We anticipate that improvements in wireless drivers and hardware will remedy this need in the near future.

Linksys WRT54G Wireless Router

While not a mobile platform, the WRT54G was, in theory, an ideal platform for Panoply. A wireless router is a wonderful place to position Panoply functionality for managing both location and communication spheres. Two JVMs claimed to support this platform—Kaffe [Kaffe] and SableVM [SableVM]. SableVM, in particular, is optimized for memory efficiency, an absolute necessity when dealing with memory-constrained platforms such as the WRT54G.

SableVM proved not to support the security libraries necessary for Panoply, nor did it possess the appropriate low-level network control primitives we use to query and control the device's network interfaces. While it would have been possible to remove all security features and cryptographic protocols from Panoply, this would have been unrealistic and run counter to Panoply's security goals.

Using Kaffe, much of Panoply does run on the WRT54G, but Kaffe on the WRT54G exhibited serialization identifier inconsistencies with array primitives that prevented interaction with any other Java Virtual Machine. The result was that serialized data structures could not be transferred between devices. As our event communication model relies upon serialization, this was a major difficulty. We spent time debugging this, but ultimately failed to resolve the inconsistencies, which lay deep inside GNU classpath [Classpath], an open-source implementation of the core Java libraries. This problem may be fixed via new releases. In addition to the serialization problem, the JIT-version of Kaffe could not be successfully built for OpenWRT. The interpreted-only version did build, but as with the Nokia 770, the lack of an accelerated JVM resulted in extremely poor performance.

4.2 Panoply Applications

In addition to the three large applications discussed later in this dissertation, we have also implemented a number of small applications and services with the Panoply middleware.

4.2.1 Context-Scoped Media Control

This application was one of the very first applications developed for Panoply. The goal of this application was to provide an intelligent application for controlling environmental media, without requiring explicit addressing of control events. We wanted to provide the ability to control environmental music, allowing the user to start, stop, skip forward, and skip backward in an audio playlist using controls on his or her local device, regardless of where the actual media application was executing.

This application consists of two components, the Media Controller, and the Media Remote.

MediaController

The MediaController application executes in the context of the device sphere running on a computer with a soundcard and attached speakers. It is implemented in 100 lines of Java. It registers with the local sphere for events of application type *MediaControl*, including events with subtype START, STOP, FORWARD, and BACK. When the MediaController receives a *MediaControl* event, it executes a local system call to communicate with the local media player. This allows the MediaController to start and stop music playback, as well as skip forward or backward in the playlist, as encoded in the received event.

MediaRemote

The MediaRemote is a simple Panoply application that provides a command-line interface to the user, and requires only 110 lines of Java. The user can issue commands such as “start,” “stop,” etc., that the MediaRemote turns into a corresponding *MediaControl* event and injects into the local device sphere. These events are attached to a *single-path scope*, which directs the event to a single-event receiver that is subscribed to *MediaControl* events. If a media controller is running on the local device, the local media controller would be the preferred receiver for the control events. If not, then the MediaRemote’s control events will be propagated to an interested MediaController.

The Media Control application is a small application illustrating event flow and scoping as well as shared control of an environmental application. Multiple individuals are able to control an environmental music service through their personal devices using the MediaRemote application; if they are also running a local MediaController, MediaRemote events are delivered appropriately.

In the event that there are multiple MediaController applications, the current MediaRemote cannot properly distinguish between them, and the control event would be delivered indiscriminately to one of the MediaControllers. There are a number of ways to work around or fix this, including a *type-following scope* that would limit event propagation to location spheres or a *time-to-live scope* that would limit the event propagation based on hop-count. Currently, Panoply does not have a notion of subscriber distance, and thus cannot talk about the “closest” interested subscriber, although this feature would be valuable and is discussed in Chapter 9, Future Work.

4.2.2 Group-Aware Door Control

We built a second small application to demonstrate policy-based control of an environmental actuator. This application is designed to control a solenoid lock that restricts access to the LASR laboratory in Boelter Hall 3564. This application had two main control interfaces. The first allows any member of the lab to unlock the door from their portable device. The second unlocks the door in response to a knock, if and only if a lab member's device sphere is also connected to the room sphere. There are three applications that comprise the Panoply Door Control system.

DoorController

The DoorController application executes in the context of the Boelter 3564 location sphere; the computer hosting the sphere is attached via a serial connection to a simple actuator that can toggle the lock solenoid. The DoorController application is implemented with 102 lines of Java. The application registers with the location sphere for events of application type *DoorControl*. When the DoorController receives one of these events, it uses the Java *RXTXcomm* [RXTX] library to signal the external actuator to either lock or unlock the door.

DoorRemote

Similar in function to the MediaRemote described in Section 4.2.1, the DoorRemote is used to manually unlock and lock a controlled door, and is implemented in 100 lines of Java. The DoorRemote can present the user with a command-line interface, allowing commands to be entered that are then turned into *DoorControl* events. Alternatively, the

DoorRemote can be used in control scripts by passing in “lock” or “unlock” parameters to the Java executable.

AcousticDoorRemote

The AcousticDoorRemote is an automated door control module that generates an unlock event in response to double- or triple-knock at the door; this module is implemented in 250 lines of Java. The AcousticDoorRemote application can be co-located with the DoorController. A microphone attached to the door provides an audio input stream which is analyzed via a fast Fourier transform (FFT). “Knocks” are registered as discrete audio events which exceed a fixed trigger threshold. Two or three knocks in quick succession (inside a specified time-window) are considered a “knock event.” Upon detecting a knock event, the AcousticDoorRemote generates an unlock event and submits it to the location sphere. Policy in the location sphere restricts the application event flow—the event is accepted if and only if a device belonging to a LASR group member, as determined by group membership vouchers, is attached to the location sphere.

Currently, *DoorControl* events are sent with a *type-following* scope that limits their propagation to location spheres. Only one door in Boelter 3564 is equipped with a DoorController, so this is appropriate. If more than one door in Boelter 3564 was equipped thusly, it would be necessary to direct control events, either encapsulating them inside a *DirectedEvent*, or through some other event scope that would ensure that events were delivered only to the associated door.

The AcousticDoorRemote’s use of policy points out a need for a notion of “user activeness,” or idleness metric. If a user leaves their device in the laboratory and goes to lunch, their device is likely still attached to the location sphere. As a result, the AcousticDoorRemote will continue to unlock the door. A notion of user idleness that could be leveraged by policy would be valuable—it would allow the policy for the DoorController to be stated in terms of participation and a required minimum user activity threshold.

Overall, Door Control is a very useful application that has been used a great deal in our laboratory. Moreover, it has been invaluable as a test application for experimenting with event flow and application policy.

4.2.3 User and Location Aware Desktop Access Control

This application, also known as the “Loco-Social Lock,” manages screen access control on any number of user devices distributed across different locations. As users move between these areas, their devices in the different locations have their screens locked and unlocked as appropriate for their current location. When the user moves out of a room, any active devices that are under the control of the user immediately have their screens locked. Additionally, when the user moves into a room where there are locked devices under his or her control, the devices are unlocked. This application was inspired, in part, by Corner and Noble’s Zero-Interaction Authentication (ZIA) [Corner2002]. In ZIA, users carry a short-range radio-frequency token that stores a cryptographic key which, in turn, is used to encrypt the keys that protect the users’ data. When the computer wants to access an encrypted file, it must request that the token decrypt the appropriate key, which can only happen if the token is within radio range. The computer also polls the token so that when it moves out of range, data in memory is re-encrypted.

LocoSocialLock Application

The LocoSocialLock application is implemented in 354 lines of Java. The application runs on every device sphere that the user wishes to influence with his or her presence in a location. This application makes use of the Panoply localization facility and a social sphere that manages the user's location context. Each participating device connects to the user social sphere, and subscribes for *LockUpdate* events. Whichever device the user carries is designated as the *lock leader*. This device publishes *LockUpdate* events when it detects a change in the user's location. Based on the location information embedded in these events, each interested device either locks or unlocks its display, dependent on whether or not the user's location matches the particular device's location.

Performance of the LocoSocialLock application is dependent upon the speed of localization. In practice, median localization time was approximately 12.4 seconds; however, this time is highly dependent upon the localization algorithm used and the selected localization maps. More localization results are available in Chapter 7.

4.2.4 Application Trace Collector

The application trace collector was developed as part of on-going research in debugging applications for ubiquitous computing. The trace collection facility is composed of two separate Panoply applications: the LogProvider and the LogCollector. The LogProvider application executes on all spheres that are being debugged. As Panoply applications execute inside an instrumented sphere, an execution trace is generated and handed to the LogProvider application. The LogCollector component runs on a node in an environment, and is intended to collect log data from any and all machines that appear in the environment. This data is then processed by an external event analysis engine.

Whenever a sphere running a LogProvider is attached to a parent sphere, it will attempt to locate and connect to a LogCollector using Panoply. If a sphere running a LogCollector is found, the LogProvider application will push all log data to the sphere running the LogCollector.

LogCollector discovery and communications between the LogProvider instances and the LogCollector use Panoply events. Connectivity awareness is handled using the sphere membership system; when the device sphere is not attached to infrastructure, no log data is published.

4.2.5 PolicyBrowser

The PolicyBrowser application assists the user in manipulating the Panoply policy database. This application allows users to observe the current state of the database, and add, remove, or modify policy, as well as load a different policy database. The PolicyBrowser application allows all of these operations to be performed dynamically on a running Panoply sphere, and does not require the sphere to be reset. The browser is a Swing-based GUI which shows all current policies belonging to the attached sphere in an editable table and provides users with operations such as "add," "remove," etc., via action buttons.

The PolicyBrowser application reflects the state of the sphere in whose context it is running, and adjusts accordingly when the sphere state and policy database change. To remain synchronized with the internal policy manager's state, the PolicyBrowser subscribes for POLICY events. This allows the PolicyBrowser to receive any and all events that the policy manager receives and generates, allowing it to remain loosely coupled with the policy manager.

4.2.6 NegotiationGUI

The Negotiation GUI application is a status visualization utility that tracks the sphere negotiation process as it occurs. The user interface represents the two negotiating spheres using a pair of vertical bars; as negotiation proceeds, arrows are drawn representing the flow of negotiation messages. Mouse-over tooltips associated with the arrows provide detailed descriptions of each negotiation step. The NegotiationGUI interacts with the local sphere in a manner similar to the PolicyBrowser application, through subscribing for POLICY events; the NegotiationGUI filters in only those events that contain negotiation data.

The NegotiationGUI application has proven to be extremely valuable as a visualization aid and debugging tool; it has revealed many issues, such as multiple simultaneous negotiation sessions with a pair of peers, as well as deadlock issues in the negotiation state machine.

4.2.7 Peer-to-Peer (P2P) File Exchange Application

This application is designed to support basic peer-to-peer file sharing between Panoply spheres. The P2P application supports keyword-based searching, transfer recovery, and request striping across multiple spheres. The P2P application was designed to demonstrate the flexibility of the Panoply architecture and a test application to explore content-based policy and negotiated agreements between spheres regarding allotted bandwidth, sharing limitations, and re-negotiation. This application consists of a client application, the P2PClient, which runs on all devices that wish to participate in the file exchange.

P2PClient

The P2PClient application extends the basic search and delivery primitives that were implemented for the Smart Party application, described in Chapter 7 of this dissertation. The basic search and delivery mechanisms used to transfer media files in the Smart Party were not sufficiently flexible for the P2PClient, so they were extended in several ways. Specifically, the P2PClient added the ability to search for files using a partial pattern, smart transfer recovery to handle interrupted sessions, and also the ability to request different pieces of a single file from different remote spheres.

The P2PClient registers for both application events and system MEMBERSHIP events. Specific application events of various types are used to support the media search and delivery protocols. More general membership context was delivered to the P2PClient via membership events. These let the application know when spheres of interest connect and disconnect from the local sphere. The actual media delivery protocol leverages Panoply DirectedEvents to send requests and content to specific spheres.

4.2.8 Summary of Applications and Utilized Features

Table 6 summaries the Panoply features utilized by our current Panoply applications, as well as the lines of code for each application. As shown in the table, applications varied widely in the set of Panoply features they used. Simple communication facilities are clearly (and obviously) the least common denominator—all of these applications are network applications and need to communicate. Scoping and membership context were also very useful and leveraged by many of the applications.

Application/Feature	Pub/Sub	Scopes	Localization Maps	Vouchers	Membership Context	Lines of Code
Smart Party	X	X	X	X	X	Master: 1250 Client: 550
Locative Media App	X	X	X	X	X	1450
P2P Client	X				X	950
Door Controller	X	X		X		Server: 90 Client: 80
Media Controller	X	X				Server: 80 Client: 80
LocoSocial Lock	X	X	X		X	350
Policy Browser	X					430
Negotiation GUI	X					420
Application Trace Collector	X				X	Collector: 450 Provider: 200

Table 6. Summary of Panoply features utilized by existing Panoply applications

Panoply clearly simplified application design and development time. The most complex Panoply application, the Smart Party, required about only two weeks of programmer time. Simpler applications, such as the Door and Media Controllers and the Loco-Social Lock were each designed, implemented, and debugged in a matter of several days apiece. In general, the various Panoply developers felt that Panoply greatly simplified the development task by providing many essential ubiquitous computing features, and by providing an asynchronous communications model appropriate to many applications within this domain.

Chapter 5

Case Study: Security for Nomadic Devices

Within this last decade, the mobile computer user became commonplace. We take computers from our home network to our office network, to our schools, to airports, to local cafés, and to countless other locations that provide wireless access to the Internet and local area network services. The near-pervasive presence of wireless networks in much of the world has made this type of network migration extremely easy. Many existing wireless devices are configured by default to access any local network that will answer a request for service.

Unfortunately, this easy access to local wireless networks and the Internet is also a huge security problem. As these users and their mobile devices migrate from network to network, they take with them vulnerable applications and services, and worse, malicious code. Typical security mechanisms are incompatible with this degree of device mobility and this scale of dynamism. The traditional security focus has been on protecting the "inside" network from the "outside" network, where there are well-defined boundaries. However, these boundaries become indistinct as users freely move devices back and forth with little thought to security.

An immediate consequence of mobility is that firewalls are rendered impotent to prevent the spread of malicious code. Vulnerabilities are exposed to many potential attackers, and malicious worms and viruses have access to many more local networks in which they can spread. The sudden infection and spread of worms, such as Blaster during August 2003, illustrate this problem. Numerous site-wide infections were caused by users bringing their infected laptops into a supposedly secure zone, allowing the worm to propagate freely [Hilley2003].

Unfortunately, this situation will only continue to escalate as more and more mobile devices and services emerge. New consumer devices and appliances such as the TiVo® and other digital video recorders, networked gaming consoles such as the Xbox®, the Xbox 360®, the Nintendo Wii®, and the Sony PS3® are already gaining first-class status as members of home computing environments. Simultaneously, network services will be offered virtually everywhere, allowing handheld and mobile devices to acquire connectivity wherever they may be taken. Millions of wireless users are already engaging in risky computing behaviors in public wireless commons, which bring together numerous mutually unknown and potentially threatening devices.

This emerging trend requires new security solutions to address its unique demands. Hosting networks need to ensure their local integrity, and ensure that new clients adhere to local policy requirements. The level of protection may differ with the type of environment and relationship between the environment and member devices. Home and business environments may proactively keep occupant and employee devices up to date with the latest security patches and virus definitions. In public areas where intrusive mechanisms are not feasible, infrastructure must still seek to mediate network association and prevent casual infection within the environment.

Panoply and the Spheres of Influence model are particularly well suited to address these security requirements through procedural policy-based configuration and mediated connection management. To demonstrate this, we implemented QED, a security service designed to protect nomadic users and the networks that host them. In three stages, devices are *quarantined, examined,* and, if necessary, *decontaminated* upon entering a new network. The quarantine stage isolates newly joined devices—potential clients—from the entire network, establishing a secure session between the visiting device and a security manager. This serves to protect the incoming device and devices within the network from each other. The device is then examined and evaluated by the security manager which determines whether the device meets local policy in terms of application and operating system patch level, offered services, or any other specified metric. If the device does not meet local policy, it may be offered restricted service or possibly none at all. Alternatively, the hosting network may offer decontamination assistance; for instance, it might provide appropriate signed software updates or suggest services to disable in order to receive service.

The original QED service was implemented with an early version of the Panoply framework. After the initial design and subsequent publication of QED, a number of commercial products emerged that address the problem space using similar approaches. These will be discussed more in Chapter 8, Related Work. While these other systems are related, QED was the first known example of a procedural network access control system that specifically addressed the unique needs of mobile devices. This chapter will discuss the design of QED, our service implementation, and will also present an evaluation of typical use cases. Additionally, we will present results illustrating the benefit of QED in slowing the spread of malicious software through mobile channels.

5.1 Motivation

As millions of wireless users migrate between home, office, coffee shop and bookstore, they move from one wireless access point to the next. They take with them not only their mobile devices, but also vulnerable and poorly configured applications. Additionally, they may also unwittingly carry digital hitchhikers picked up during their travels, in the form of viruses, Trojan Horses, denial-of-service daemons, and other such malware. As the user migrates between wireless networks, these risks threaten the integrity of the other environments, as well as that of other peers within those environments.

Consider the case of Bob, an office employee. Bob uses a laptop for work, and is reasonably competent with his computer, but is not a security expert.

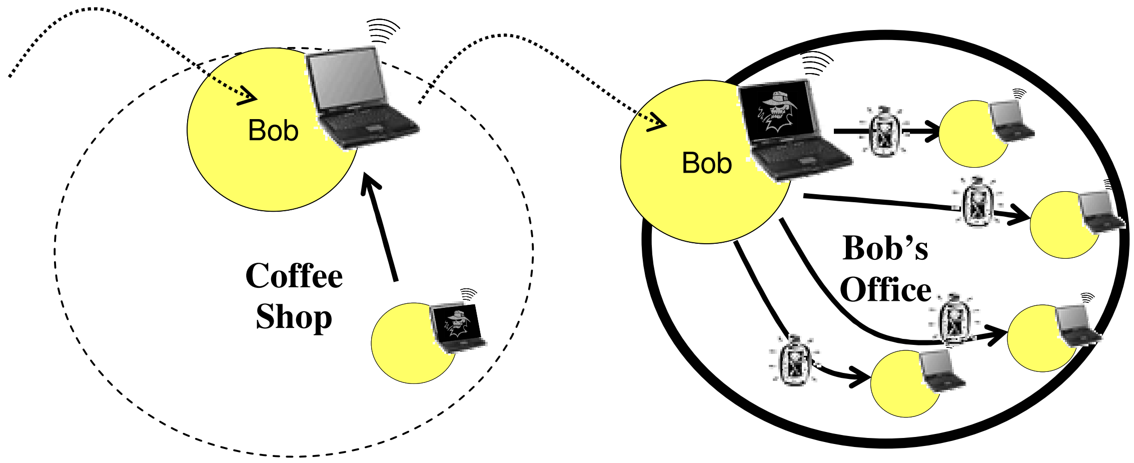


Figure 12. Spread of infection via mobile vectors

He takes his 802.11-enabled laptop to and from work daily. He also frequently interacts with other public wireless networks to check his e-mail and browse the web.

Before work one morning, Bob stops by his local coffee shop to buy a latté and read the news on his laptop. Unfortunately for Bob, the coffee shop is not a particularly secure environment. A new quick-spreading worm is out in the wild, and another patron has brought into the coffee shop a laptop that is infected with such a worm, e.g., W32.Blaster.Worm [Blaster].

In short order, Bob's laptop is attacked and infected, as illustrated in Figure 12. Oblivious to the infection, Bob takes his laptop with him to work, and connects to his corporate network, behind his company's firewall. Other machines on Bob's local LAN are vulnerable and infected as the worm spreads quickly. If any of the newly infected machines are also mobile, they can be taken out of the office and will continue to spread the infection.

This simple scenario illustrates the threat inherent in relying on traditional mechanisms to deal with nomadic users and devices. Unfortunately, this scenario is far from fiction. Occurrences very similar to this led to the infection and subsequent re-infection of the UCLA Computer Science Department network during the late summer and fall of 2003. Similar experiences were also common elsewhere [Sophos2003, UCI2003].

In a Spheres of Influence-based system, Bob's experience could be much different, as illustrated in Figure 13. Whenever Bob is at work, his laptop participates in a community of devices. This community imposes policy regarding application and service updates that must be applied to member devices in order to belong to the community and access communal services, such as Internet or LAN connectivity. When Bob brings his laptop to work, it goes through a mediated join process with the office device community. Bob's laptop is examined, and it is determined that he needs the newly released service patch. Bob's laptop examines the patch and cryptographically verifies its integrity. The patch is installed, and Bob's laptop is allowed to become a full member of the office device community.

Later, when Bob goes to get a coffee and browse the web at his local coffee shop, a compromised laptop attempts to attack Bob's laptop. However, the attack fails since Bob's laptop is no longer vulnerable.

5.2 The QED Model

The general QED model is based on the observation that when a mobile device intends to

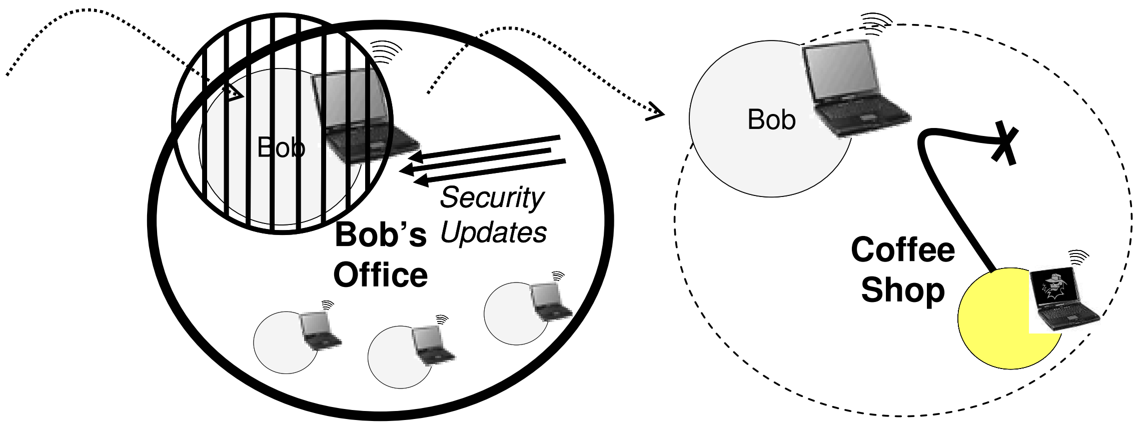


Figure 13. Spheres of Influence-based dynamic security perimeter

join a communication network, the device and the network must mutually participate in a process to determine if and how the device is going to join a network, and what the device's role will be in the new network. This must be done as securely and safely as possible, without interference from external entities. This leads to the formulation of the three distinct phases of the QED process, including:

1. A **quarantine** phase to protect the device from external interference and to protect existing network denizens from potentially hostile devices.
2. An **examination** phase that allows the device and environment to each determine if the other is acceptable.
3. A **decontamination** phase that assists with any recommended or required configuration changes.

When actualized, the implementation of these components may not be quite so distinct, i.e., there may be blurring of boundaries between examination and decontamination. However, there is a general time ordering—quarantine must begin first, followed by examination, and then followed by decontamination to ensure safety. Each of these components of the QED paradigm is discussed in detail below in terms of requirements and possible mechanisms; the actual QED implementation used for our prototype will be detailed later.

This discussion will be primarily from a network-centric perspective, i.e., the network examination and decontamination of incoming devices. However, devices also have needs and capabilities in the QED model—they maintain the right to determine network identity and capabilities, and also request needed access privileges.

5.2.1 Quarantine

Quarantine refers to securing the connection establishment process and restricting peer network communication to the necessary minimum. From a device perspective, this means denying incoming messages from network devices other than an authorized security manager for the network the device is attempting to join. From the network perspective, this means isolating an incoming device from other devices on the local network and the Internet.

Every environment must have a security manager, ideally located at the gateway of the network, where it can enforce two types of isolation. The first type of isolation involves protecting local machines from the outside world. All communication goes through this manager. This type of isolation is provided today to varying degrees by firewalls. The second type of isolation protects local peers within the network from the entering device. This capability is at least as important as isolation from the outside world, and rarely provided by traditional security models.

Quarantine requires establishing a secure channel between the network security manager and the device. This typically requires the establishment of trust, via credentials such as certificates or Panoply vouchers, provisioned either before the fact, or via a network enrollment procedure. Once credentials have been presented, both network and device must determine if they wish to continue the procedure based upon the nature and authority of the presented credentials. The policy that guides this decision may be device- or network-specific; for the purposes of this discussion, we assume that the process continues and both sides are satisfied with the presented credentials.

While this secure channel is being created, the potential client denies all incoming traffic not associated with the creation of this channel. For example, if using an IPsec-based VPN with X.509 certificates, we only allow IPsec packets related to the creation of the IPsec security association and the associated certificates to be exchanged. Similarly, if using TLS over TCP-based tunnels, local firewall filters are used to drop all traffic that is not part of the TLS session with the security manager. After the secure channel to the security manager has been created, the client denies all traffic not part of the channel, preventing attacks from other devices in the network.

A potential client will remain in quarantine during the entire QED process. It may in fact be desirable to maintain a partial quarantine indefinitely, depending on the degree of trust between client and network. There are many ways in which an entering device can be isolated from the rest of the network. It can be explicitly prohibited from communicating with other devices within the network; likewise, existing network members can be prohibited from communicating with any device that has yet to be accepted into the network through the use of a white list—we refer to this practice as *network shunning*. White listing in this context scales because the white list only needs to include accepted members within the local IP subnet; existing security infrastructure, e.g., an enterprise firewall, etc., handles inter-network protection.

One other ongoing quarantine task is identifying when devices leave and rejoin the local network. Depending on network policy, the rejoining device may go through the same process as others, or a more limited form of QED based on length of absence, etc. Group membership management may be performed through beacons, keep-alive messages, or other similar techniques.

5.2.2 Examination

Once quarantine has been established and the prospective client device and network have established a secure channel, the examination phase can begin. During examination, the network assesses the incoming device and determines if the device satisfies or can satisfy entry requirements, as specified by a local policy. Similarly, the device can potentially examine the network and determine if the network meets the device's needs in terms of offered services and network capabilities, security constraints, and so on.

Examination is, in essence, a limited version of the more general Panoply policy negotiation model. Bi-directional requests and responses are used to determine whether the negotiating entities wish to engage in a network relationship. Many different mechanisms can be incorporated into an examination procedure, including package management tools, network port scanners, virus scanners, and configuration analysis tools.

Examination of the Supplicant by the Security Manager

The complete and exhaustive examination of the contents of a supplicant device at the time of entry may often be unrealistic. Depending on device capacity, it is often infeasible to perform such an examination in a timely manner. The point of supplicant examination is not to duplicate the efforts of a virus or malware scanner in whole, but rather try to determine whether or not the device is considered sufficiently safe to operate in the network in question. This can include requesting and receiving cryptographic attestations as to the patch level of various operating system components and system services, as well as determining the last time the device had been scanned for viruses. It may also include a port scan and service fingerprinting that seeks to identify vulnerable or undesirable services.

These examples illustrate the different modes of examination. Examinations may be *external* or *internal*, and additionally, they may also be *immediate* or *time-windowed*.

An *external* exam refers to external probing or monitoring that is performed to analyze the device. This may include traffic monitoring, port scanning, service fingerprinting, and so forth. This is typically non-invasive, and can be performed entirely by the security manager.

Internal examinations refer to an examination in which the device voluntarily cooperates by executing appropriate software. This mode of examination may include virus scanners, package examination, configuration examination, and so on. These results are, in general, generated on the device in question, and in general should be secured via trusted platform management [TCPA] or through another form of secure code execution. Since an internal exam requires the device to run code possibly provided by the security manager, safeguards must be used to ensure the integrity of the examination code and to protect the device from malicious examination attempts. Cryptographically signed examination modules, signed by a trusted third party or by the network in question, can be verified for authenticity. Once authenticated, the device must consult its own policy to determine whether or not to execute the module. Other techniques, such as proof-carrying code [Necula1997,Sekar2003], can be used to certify module functionality and safety.

Immediate exams and *windowed* exams differ in their time requirement. Immediate exams must be based on results generated during the current QED session. A windowed exam refers to an exam that accepts results generated anytime within a specified time window. An example would be a network requiring a daily virus scan. If a device already went through a full virus scan and can present a valid cryptographic attestation indicating a clean bill of health, it can forgo further scans that day.

These examinations are by their nature limited in scope and correctness. To that end, local policy must assume a threat model in which the results of examination are a guideline, but cannot be guaranteed. As a result, the QED model advocates maintaining partial quarantine by restricting communication to that which is absolutely necessary, as well as continued monitoring of all network peers. If problems are later detected, the device can be placed back into full quarantine and further examined, or ejected from the network. Additionally, when security events occur, e.g., a CERT advisory is issued, or a new system security update is available, devices may be required to resubmit to the QED process.

A characteristic of ubiquitous computing environments is the large degree of heterogeneity in operating system configurations and applications, both in the client devices and the environments themselves. A given environment may not possess sufficient knowledge of a potential client to perform certain types of examinations, such as package scans, etc. Certain basic types of examinations will typically still be applicable—port scans, service fingerprints, external traffic monitoring, etc.

Typically though, more invasive examination is generally needed only in areas where it is easier to perform. Internal examination including a virus scan and package examination would typically be required in environments such as the home or the office where the device and the network have a pre-existing relationship. Within one's home or one's office, it is quite appropriate to require a much more stringent examination and decontamination of entering devices. It would be perfectly reasonable for a business to require that their employee's devices undergo a rigorous scan before being allowed onto the network; similarly, within one's own home, the network and the device have a pre-established trust relationship. The network would know the recent state of the device, greatly simplifying the task of detecting changes to the operating configuration.

In public environments, such as the coffee shop Bob visits in our introductory scenario, it is not reasonable to expect the user's device to accept intrusive examination or decontamination. External scans, monitoring, and basic quarantine may be all that are acceptable to the user. However, this is sufficient to reduce the casual spread of many forms of malicious code.

5.2.3 Decontamination

The decontamination component of QED begins after the initial examination is complete. Decontamination attempts to bring the target inline with appropriate local policy. From a network perspective, this means updating a device's packages, disabling or disallowing local access to some services, requiring configuration changes, or removing malicious code. From the perspective of the supplicant device, it may include asking the network to run a needed service, alter the configuration of an existing service, or allow a given service or protocol. In general, decontamination is a negotiated process. Each party takes steps that it is willing to undergo to achieve its particular goals. One party does not have the power to force the other side to do anything which is against the other's local policy.

Decontamination can take many different forms. Virus scanners can automatically destroy or quarantine viruses. Package management tools can apply new security patches. The security manager can instruct the client to stop certain services. Again, any software locally executed on the client must be cryptographically signed by a trusted authority before the client device will allow it to run, and techniques used to verify the safety and authenticity of mobile code can be leveraged to offer further protection.

When decontamination is complete, the client is given privileges to communicate and interact as appropriate—the degree of interaction allowed should depend on the needs and identity of the device.

Time is a serious constraint in decontamination. For this process to be feasible in real environments, it must be performed as quickly as possible. This requires that the security infrastructure actively and aggressively cache the latest anti-virus and software updates. Work done in web caching and content distribution can be leveraged for this purpose. Profiles of what devices use the network at a particular time and what applications and services are likely to run could be maintained to assist with the update caching process. If a particular update is not found in the cache, it must be fetched from the appropriate source through the Internet, adding additional overhead to the process. Lastly, the decontamination process has a complex relationship with examination. These processes may partially overlap or interleave since their functions are so closely related—as a result, necessary packages can be proactively cached as soon as their need is determined. Care must be taken to avoid potential denial-of-service scenarios possible through this interleaving of functionality.

5.3 A QED Service

The first Panoply prototype was designed primarily as a QED service within the LASR environment. Our target devices included Linux-based laptops equipped with 802.11b wireless network cards. The framework was designed to provide wireless service and security updates to several dozen wireless clients, while keeping unknown or vulnerable machines in a restricted quarantine.

The center of the QED service is the network security manager. This Linux-based desktop acts as a wireless access point and router for the QED subnet. Additionally, the security manager offers authenticated DHCP and IPsec VPN services.

5.3.1 Network Discovery, Configuration, and Quarantine

The QED client software configures the client's wireless interface to monitor all wireless traffic, listening for QED beacons. The security manager for each network broadcasts wireless beacons that advertise the presence of the QED-enabled network. These messages include a description of the network, the network's public key, and the IP address of the local security manager. QED beacons are encoded as UDP broadcast packets. In order for the client to observe the beacon, the QED client uses JPCAP [Jpcap], a Java interface to *libpcap* [Libpcap], to analyze relevant UDP broadcast and identify QED beacons.

When a client overhears a QED beacon, it must decide if it wishes to learn more about the network and possibly attempt to join. For our prototype, laptops operating in the environment were given X.509 certificates for the appropriate Computer Science Department networks; the device must verify the beacon against its set of certificates to determine its validity. Given the current Panoply implementation, one way to extend this and improve the scalability would be to use Panoply enrollment support to offer a mechanism by which client devices could acquire valid certificates for their local environment.

Once a client elects to join a network and is associated with it, it uses the Dynamic Host Configuration Protocol (DHCP) to obtain all the network configuration information that is necessary in order to become a part of the network and to be able to communicate with the other members. The DHCP protocol [RFC2131] is primarily used to assign a dynamic IP address automatically to a machine that is expected to be a temporary member of a network. The protocol relies on communication between a DHCP server hosted by the network and the DHCP client on any device that wishes to connect to that network.

For our QED prototype, we use DHCP as a means of assigning an IP address to a client, as well as disseminating other configuration information. We have implemented a modified version of DHCP that allows clients and servers to authenticate each other. In a DHCP packet, all configuration information is stored in options [RFC2132], one of which is an authentication option [RFC3118]. RFC 3118 provides general guidelines for transmission of authentication information.

Our modified DHCP achieves three things: mutual authentication of client and server, establishment of DHCP session keys, and registration of the client's RSA public key, used for IPsec. The protocol works as follows:

- § The client first broadcasts a DHCPDISCOVER message. Included in the message is an authentication option that includes the client's RSA public key. The message and authentication option are signed using the client's private key. The client will sign all future communication in this session with its private key.
- § The server verifies the message and the client key. It also caches the client's public key for later use in authenticating client messages. At this point, the client's key may be optionally verified against an access control list.
- § The server replies with a DHCPOFFER message that offers the client a specified local IP address. The authentication option contains a public key that the server has previously generated—when possible, this should be consistent with the network key advertised in the beacon. The server will use its corresponding private key to sign all messages belonging to the current session.
- § The client caches the server's public key. The client sends out a signed DHCPREQUEST message. This message requests the IP address offered to the client.

The protocol finishes when the server answers with a DHCPACK message, again signed with the appropriate session keys. In our prototype, we used the SHA1 algorithm for signing and verification, with 1024-bit keys.

Additionally, when the DHCP server registers the RSA public key of the client, it performs a secure DNS update to publish the acquired key into the local DNS database. The key is mapped to the newly allocated IP address given to the client.

After successfully acquiring an IP address via DHCP, the client initiates an IPsec tunnel to the security manager, using the RSA public key for the security manager acquired from the network beacon. The IPsec implementation on the security manager uses the client key, accessed via local DNS, and the tunnel is established. The client routes all traffic through the tunnel, denying all incoming traffic that does not originate from within the tunnel. The potential for attack between DHCP address setup and IPsec VPN establishment can be reduced by only accepting inbound IPsec SA establishment packets from the security manager, as identified by the network beacon. Absent vulnerabilities in the IPsec SA setup mechanism, this will prevent attacks issued during this window.

Meanwhile, the security manager is aware of the new potential client. However, no outbound communication is allowed, nor is any other local device allowed to address packets to the quarantined client. The client must successfully complete examination and decontamination before any further interaction is allowed.

Network membership is managed via periodic heartbeats from the gateway and clients. If the QED security manager does not hear a heartbeat within a configurable time period, the IPsec SA is torn down, and no further traffic is accepted. The client must then reestablish the IPsec SA, and potentially be reexamined. This model is similar to that used in the current Panoply implementation.

5.3.2 Examination and Decontamination

The QED prototype offers an easily extensible examination and decontamination facility. Since the particular mechanisms for these processes are network and policy dependent, the implementation takes a modular approach. The QED framework manages the control flow of examination and decontamination, but the actual processes are carried out by a suite of plug-ins. Network policy, described in a configuration file, indicates which set of plug-in components should be executed on the client device. The framework and the associated modules are written in Java. QED modules rely on Linux system tools, including *rpm*, *nmap*, and a few custom scripts. In our prototype, the examination and decontamination phases largely overlap due to the modular nature of the implementation.

The Examination Framework

The QED framework is divided into two parts: the code that runs on the security manager—the examiner—and the code that runs on a client—the target. The framework code on the examiner reads a configuration file that specifies which modules to run in what order for each supported Linux distribution. The configuration file can also specify arguments to pass to modules. When the IPsec VPN with the potential client is established, the security manager initiates examination.

Once examination is initiated, the client device's operating system version is determined by querying the examination daemon running on the device. Once the examiner determines which modules need to be executed, the information is sent to the target, which checks its local module cache. If a module already exists in the target's cache, its MD5 checksum is compared to that of the same module on the examiner. If the MD5 checksum matches, the cached copy is loaded and executed; if it does not match, the cached copy of the module is overwritten by an updated version transferred from the examiner. Examination modules can additionally be signed by the network, or by an accepted third-party signatory.

In our environment, the user is given complete discretion regarding what code is executed on his device. Therefore, the framework allows the target system to query locally cached modules for information describing their functionality. Using this information, the user can determine if it is acceptable for the modules to run. The user can also take into account the time for each module to run and the amount of privacy maintained by the module. Depending on the user's decision, the framework then runs the module or skips to the next one. This process can also run in an expedited mode where approved modules execute automatically.

Pluggable QED Modules

The modules, like the framework, can be separated into two categories: code that runs on the examiner and code that runs on the target. The two parts communicate with each other to accomplish the various examination and decontamination tasks. The modules can be written to be generic or specific to an operating system or distribution. An individual module can also be written to handle both examination and decontamination.

Before a module runs, it must provide information about its functionality to the user through the framework. After a module terminates execution, it returns its status to the framework. Using this status, the module can control the flow of the examination and decontamination session by specifying other modules to run. For example, a module can be designed to check for a very specific virus. If the target is infected with that virus, the module can instruct the framework to load a separate module that will remove the virus. Once this decontamination module removes the virus, the framework loads the next module as indicated in the configuration file.

We have implemented the following modules:

RPM Check

This module examines the target machine to see if any of the installed packages are out of date. It uses a local archive of update packages that are automatically synchronized with the appropriate update directories by the security manager. The QED prototype supported Red Hat Linux 7.3, 9.0, and Fedora Core.

The examiner sends a list of package names to the target. The target module collects the relevant information from its local RPM database and sends it back to the examiner. The examiner compares this information against the packages stored in the local archive. If any of the target packages are out of date, the examiner will indicate this to the target and then instruct the framework to run the RPM Update module.

RPM Update

This module is only run if the RPM Check module determines that there are packages on the target that require security updates. It expects a list of packages to update from the RPM Check module. The new packages are transferred to the target and the *rpm* tool is used to update the target. We rely on the package manager's dependency checking to resolve conflicts.

Both the RPM Check and RPM Update module must, for the most part, trust the target machine to properly execute the modules and return the results.

Port Scan and Service Blocking

This module relies on the *nmap* tool [Nmap] to scan the target to identify which services are running. The examiner initiates the scan and compares the results to a pre-specified policy to determine whether the target is running any services that are disallowed by local policy. Using this information, the examiner instructs the target device to block all incoming traffic on unacceptable ports using *iptables* firewall rules [IPtables]. Additionally, the examiner can block routing for the given ports, not allowing any other computers to access the blocked ports. The examiner then rescans the target to check that the rules have been put into place.

Since the port scan is handled external to the target, the results can be reasonably trusted—at least at the moment they are acquired.

File Property Check

This module is essentially an integrity check on the most commonly used executables to check for signs of tampering or the presence of Trojan horses. It uses the file information embedded in the RPM package archives; examples of pertinent information include file location, file permissions, and MD5 checksums. The examiner requests information for specific files from the target. It then extracts the same information from the local archive of RPMs and compares it to this. If inconsistencies are found, they can be dealt with in different ways, depending on policy and file type. It is important to note that this type of an examination is less useful to those who frequently rebuild their systems and apply custom patches. For most user systems, however, binary executables change infrequently, and inconsistencies may be a cause to alert a system administrator. Temporary fixes are possible, such as archiving the altered files and using the RPM Update module to overwrite the altered file(s). It is important to be careful and avoid inadvertently destroying user data.

This module can also compare current system state to a recorded baseline state, and identify recent changes. This would be very useful for one's home or office environment because logs of the contents of one's machine when it last visited the environment can be maintained and unauthorized changes to the system contents can be detected.

For all of these different examinations, it is important to note that they represent snapshot examinations of system state. In a full-fledged QED environment, ongoing examination would also be performed. In ongoing examination, a periodic reexamination of system state would be required, as well as environmental monitoring to detect worm transmission or other undesirable behavior.

5.4 QED Overhead Measurements

We measured the performance of our QED prototype in order to evaluate the impact of QED in terms of added network overhead. Specifically, we measured the various stages involved in joining and becoming an active member in a QED-enabled network. Within the LASR network, QED is principally used to determine the current version of installed packages and identify externally offered services. If packages are found to be out of date, updates are provided to the device. Similarly, if an undesirable network-based service is running, the device is instructed to firewall off the appropriate port.

These experiments involved two nodes. The first node consisted of a 1.3GHz AMD Athlon equipped with 512 Mb of RAM. This node was the QED client, operating on top of RedHat Linux 9.0, with kernel version 2.4.20. QED executed under the Sun JDK 1.4.2. The second node was the security manager, and consisted of a 2.53 GHz Intel Pentium 4 equipped with 512 MB of RAM and running RedHat Linux 7.3 and kernel version 2.4.18. The implementation of IPsec used was Linux FreeS/WAN U2.04. The experiment was designed to mimic the behavior of a device joining the 802.11b wireless network. The DHCP implementation used in these measurements was ISC DHCP v3.0.1rc11, and the RPM implementation was RPM v4.2.

When the client device saw a beacon announcing the QED-enabled wireless network, it automatically joined the network, getting a DHCP address and initiating an IPsec security association (SA) to the security manager. It is important to note that the DHCP caches are cleaned out after each experimental trial, so the existing leases are destroyed. This has the effect of forcing the DHCP client to go through the full four-way handshake. The security manager then initiates the examination process across the SA. Decontamination, if necessary, is performed after examination, and then the client is given full network access. Tests were run hundreds of times, and all results are reported with 99% confidence intervals.

An very typical use case is one where the examined device is current with respect to security patches—i.e., no package update is necessary. In this case, the measured mean total overhead involved in joining the wireless network is $6.6 \pm .5$ seconds. QED's operation can be broken down into three parts—the time to acquire an IP address via DHCP, the time to establish an IPsec SA, and the time to transfer the current installed package list to the security manager and examine it. Absent QED, devices would typically experience only the DHCP overhead when joining a typical wireless network. Establishing the SA and the package examination incurs 3.2 ± 0.01 seconds of additional delay before the device receives network access. Figure 14 breaks down the connection overhead into its various components.

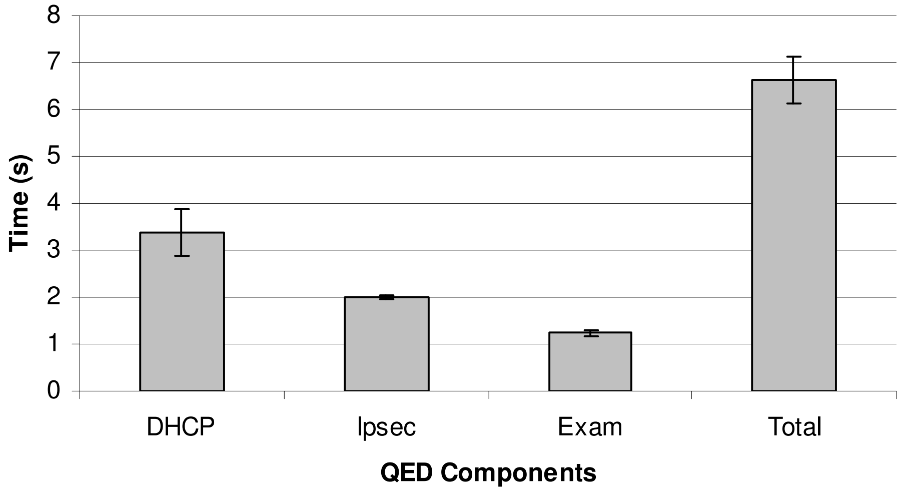


Figure 14 Mean QED overhead measurements for an up-to-date machine.
(confidence intervals are reported at 99%)

Experiments were also conducted to measure the incurred overhead when system updates were required. To observe the range of possible delays introduced by the QED package examiner and decontamination modules, we compared five different cases. We measured the null case (where no updates were necessary) and individual updates of four different representative RPM files, each with very different characteristics. The null case is the typical user experience when his or her machine is up to date.

	Exam Only	pam_smb-1.1.6	gdm-2.4.1.3	foomatic-2.0.2	perl-5.8.0-88.3
Elapsed time to examine and install (in seconds)	1.24±0.01	1.88±0.01	13.14±0.73	23.90±0.41	91.14±3.66
RPM size: (in Kbytes)		32	1,466	1,379	14,143
# of files:		8	122	1875	2560
Package Density (files/Kbyte)		0.25	.08	1.36	0.18
Uncompressed file size (in Kbytes)		68	3,962	19,373	34,123
Compression ratio		2.1:1	2.7:1	14:1	2.4:1

Table 7. Comparison of QED examination for five different cases

If an available update is found, our QED prototype initiates a transfer of the appropriate RPM to the client device. To simulate this, we forced downgrades of four different packages and then measured the time to perform QED, allowing the security manager to update the packages in question. Table 7 shows the packages selected, and presents a breakdown of the time to install each package and individual package information. The packages chosen are fairly representative of typical RPMs that might be updated. Both ends of the size spectrum were represented. Linear regression on this data indicates that package installation time is proportional to the size of the compressed package.

Additional examination benchmarks include nmap scans of incoming client devices. Measurements demonstrate that the security manager can perform a brief port scan of 200+ commonly used ports in 1.15±0.3 seconds. This can typically be performed safely in parallel with other active examination modules.

These experiments have shown that in most cases, QED imposes minimal overhead upon entering devices. Typical security updates of 1 to 2 Mb could be downloaded and installed in 10 to 20 seconds. When no security updates are required, users experience on average approximately three seconds of added delay. Longer delays are possible, however they should be infrequent, and typically only required when large packages are updated.

5.5 QED's Effectiveness at Slowing Contagion

In addition to understanding the typical imposed overhead of the QED, we also need to consider the effectiveness of QED in slowing the spread of malicious software. To that end, using wireless trace data from a large wireless deployment at Dartmouth College (available from the CRAWDAD repository [Crawdad]) we studied the spread of an idealized worm that propagated primarily via mobile contact, in a manner similar to that discussed in our motivating example and previously depicted in
Figure 12. Full details of this study are available in [Anderson2005], but we will summarize the most relevant results here.

For the purposes of this study, we assume a simple model of worm activity. The worm only propagates across wireless networks. Whenever an infected host successfully associates with a new subnet, it attempts to infect any other nodes already associated with that subnet. Access points themselves cannot be infected. In our analysis, we assume that all other devices in the simulation are susceptible to the exploit the worm uses to attack new machines, and that once a machine is infected, it continues to try to spread the infection until the machine is disinfected and patched.

The wireless trace data tracked 6630 wireless users for nearly four months in late 2003 (the Fall 2003 semester), and recorded their presence and length of stay in wireless

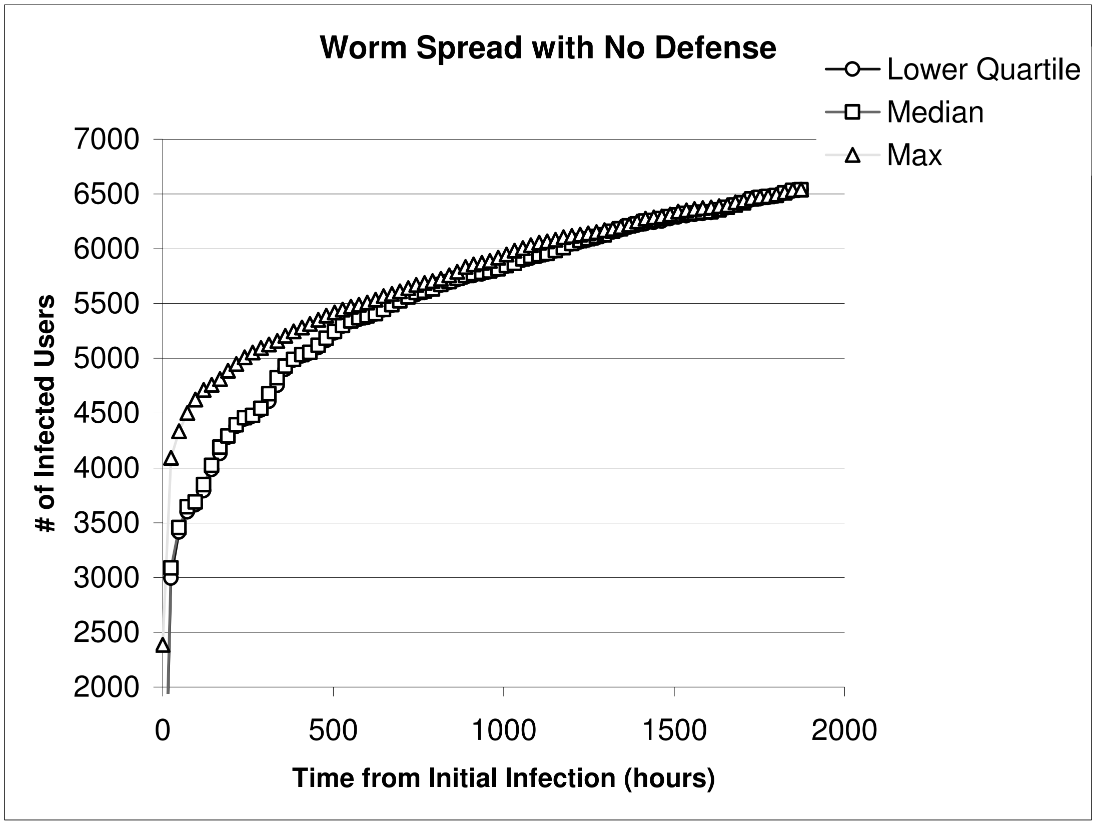


Figure 15. Summary of worm infection over time with no defense mechanisms

subnets distributed throughout the Dartmouth campus. For each of the 6630 users in the trace, we simulated what would happen if that user began the semester infected with an idealized worm that required 30 seconds to scan and infect the other machines on the subnet, and was capable of infecting any uninfected target.

Figure 15 shows the resulting maximum, median, and lower quartile curves of the number of infected hosts over time. Please note that the x-axis has been normalized such that 0 is the first appearance of the initial infecting user. This allows us to compare infection rates across initial infectors. The normalized curves show that the idealized worm exhibits infection behavior similar to the standard infection model, with a rapid jump in victims followed by a smoothing out as the worm finds fewer and fewer new targets. In the median, more than half of the users are infected within the first three days.

Simulating QED

To evaluate the effective of QED at controlling the spread of the worm, we simulated QED deployment across the Dartmouth network. To model partial deployment—a realistic setting—we simulated a random 25% deployment of QED across the various subnets, using the same worm propagation model described previously.

It is important to note that as we replay recorded trace log data in hopes of having an accurate user mobility model, we are ignoring the possibility that the users would change their network usage when infected or when interacting with a QED-enabled network. For instance, if QED was only partially deployed, infected users might decide to move to a location that did allow network access.

The base QED model used for the simulation is as follows:

Users who connect to the network are quarantined until they prove that they are up to date with the latest patches and are uninfected. Infected users are disinfected and unpatched users are patched appropriately to prevent future infection. In our simulations, we assumed 10 seconds are required to disinfect and patch an infected user. Infected users who leave the network before the time is up are not disinfected. Thus, in most cases an infected machine will be disinfected if it ever visits a location where the defense is deployed.

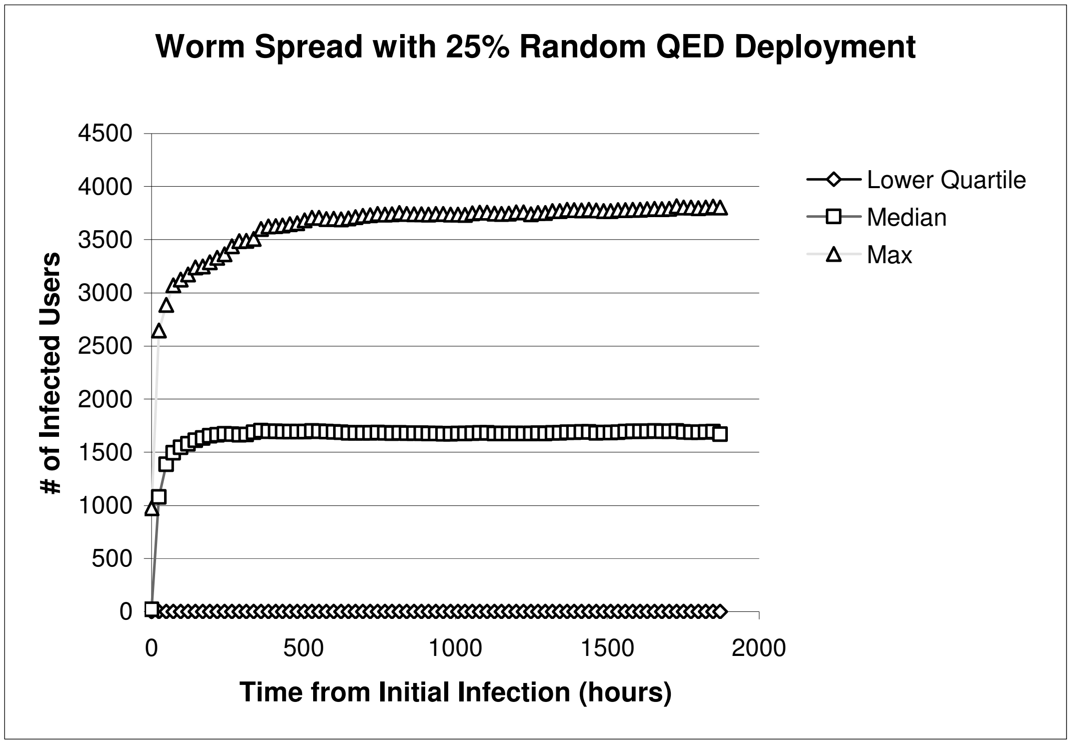


Figure 16. Summary of worm infection over time with 25% QED

Simulation results, show in

Figure 16, indicate that QED performed well in reducing the infected population.

In the median case, QED constrained the number of users infected at the end of the simulation to 22% of the population. Additionally, in 27% of the simulation runs, QED was able to contain the threat to less than 1% of the population. Finally, QED reduced the number of users ever infected to 51% in the median case. These results are good, but indicate that even a strong and aggressive defense of this kind can frequently permit a high degree of infection if its deployment is limited to 25% of all systems. Clearly, higher degrees of deployment will be required to achieve greater protection, rather than merely more sophisticated defenses at a limited number of locations. Similar experiments with varying degrees of QED deployment (Table 8) bear this out.

% QED Deployment	Median Infected Population
10%	55%
25%	22%
50%	2%
75%	1%

Table 8. Median levels of infection with varying degrees of QED deployment

Proactive Patching and QED

As most popular consumer operating systems now provide automatic warnings regarding new vulnerabilities and new software updates, and have also streamlined the system update process, we expect more users will keep their systems up to date. To examine the impact of this effect, we used the following schedule for patch installation starting at the beginning of the trace data: 50% of users patch some time within the first seven days, 25% in the next seven days, 10% in the third seven, and finally 15% will never patch on their own. Users were allowed to patch even if they were not on the wireless network, since we generally assume that they have other sources for network connectivity.

If users patch their own systems according to the schedule we devised, it can have a significant effect. However, the worm is still fast enough to infect a large percentage of the population. In fewer than 1% of the simulations, the worm was contained to 1% or less of the user base, and in the median case, 63% of the users were infected, though only 15% were still infected by the end of the simulation due to the patching schedule. Figure 17 shows the effect of proactive patching, as well as proactive patching with QED.

The combination of proactive patching with QED results in significantly better performance than either pro-active patching or QED alone. When combined, the median number of infected users is reduced to less than 4% of the total population, and even in the worst case, 9% of the population is affected. Figure 18 shows the combined median results from all of our simulations. This clearly illustrates the power of the QED paradigm in controlling the spread of malicious software, particularly when combined with reasonable patching practices.

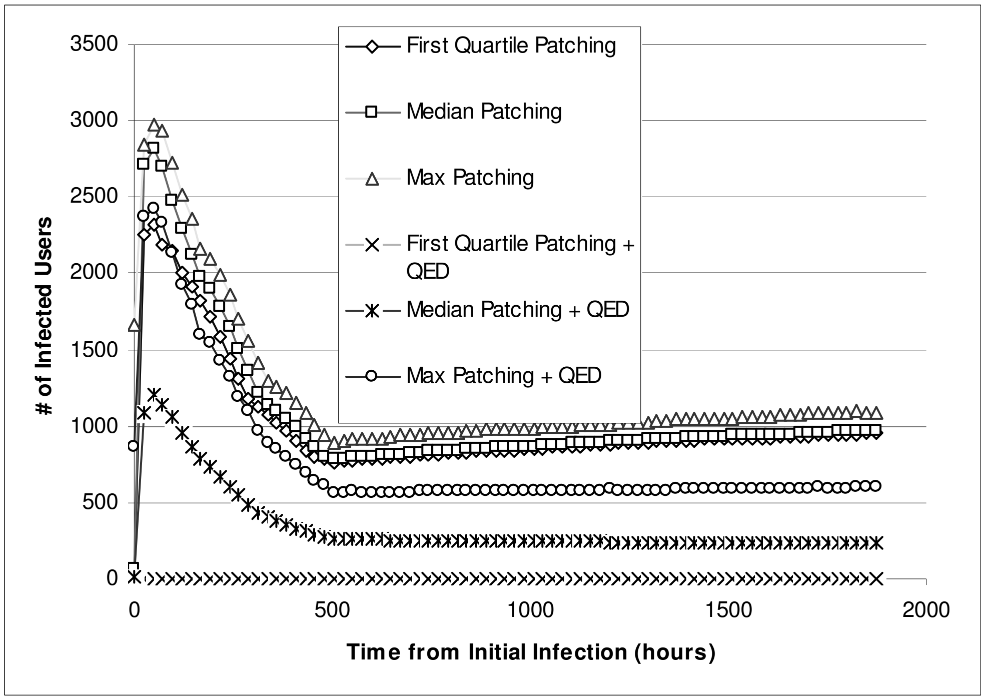


Figure 17. Worm propagation behavior with proactive patching

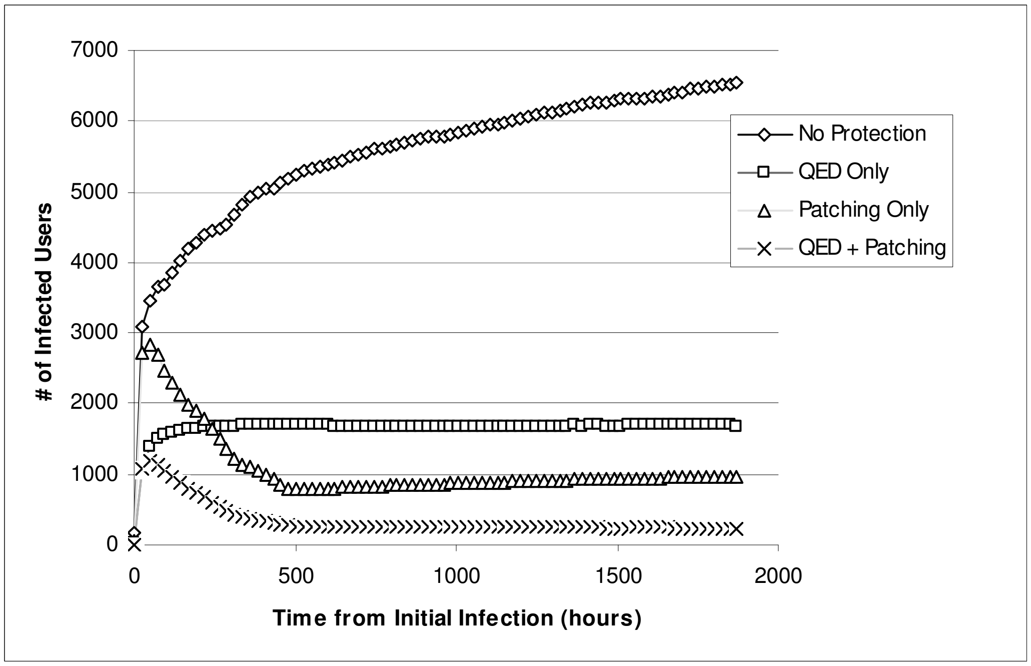


Figure 18. Comparison of median results from all worm simulations

5.6 Future Work

There are a number of important issues within QED that remain for future exploration, including trust, privacy, and scale.

Trust Issues Inherent in Device Participation

There are substantial trust issues that must be dealt with in QED. The most obvious issue involves requiring potential clients to execute examination and decontamination modules provided by a network's security manager. Modules may potentially be malicious. Alternately, devices or malicious software may lie or subvert the results of exams. The problem of ensuring the integrity of an operation is an old one. It has been investigated in the context of virus scanners, anti-piracy mechanisms, and digital rights management (DRM).

We believe that the situation is not as dire as it sounds. Given currently available systems and technology, QED can be a valuable tool. For example, had QED been available at the times of their release, our existing model would have protected against Code Red, Code Red II, Slammer, Blaster and the vast majority of known viruses that depend on exploiting well-known vulnerabilities. QED's active management of patches and system updates could also greatly reduce the vulnerability of user devices to many popular botnet packages. Additionally, in its ongoing examination mode, QED may be able to detect and stop botnet propagation mechanisms, as well as the denial-of-service activities of many popular DDOS tools in their most common modes of operation. Despite the natural limitations of examination and decontamination, system security can be greatly improved through proactive device management and through the use of aggressive quarantine. To that end, one view of QED is that of an intrusion tolerance tool that increases the overall preparation and resilience of user devices, reducing the likelihood of infection or compromise.

QED has the potential to be much more powerful as trusted computing hardware becomes more widespread. The use of a trusted computing platform such as TCPA beneath QED would allow numerous enhancements. The security manager could be certain of the fact that the client is not making unauthorized communication. Trusted modules and operating systems could not lie or otherwise subvert examination results or the decontamination process. Simply widely deploying TCPA is not sufficient, unfortunately. A great deal of work is still required to build secure examination components, and the underlying operating system components upon which we would rely.

Privacy Issues

There is a fundamental tradeoff between the ability to examine machines and the privacy desired by users. The more invasive the examination procedure is, the better the guarantee of security. The tradeoff itself could provide a solution to the problem. Networks could define various levels of access. Devices could choose limited sets of examinations they are willing to undergo in exchange for limited access. The network will always have the right to deny access to a device if necessary, and the user will always have the right to decline to submit to a particular investigation, at the cost of not receiving access.

With TCPA-based hardware, we foresee a solution to the privacy problem. A device could provide a cryptographic guarantee of the absence of vulnerabilities or malicious code that the security manager could trust, thereby not compromising security and also preserving the privacy of the device's contents.

Scalability

With an ever-increasing number of heterogeneous mobile devices entering the marketplace, the number of possible attack vectors is increasing dramatically. Much as we would like to have individual security managers store all the information about all possible platforms, it is infeasible for a practical system due to scale; the storage space at local QED sites is inherently limited. Instead, security managers could maintain profiles of the more widely used platforms, especially those that are encountered frequently in the local network, and store relevant application patches and anti-virus software. If an unknown device comes along, the security manager must procure the necessary information through the Internet. More practically, enterprise deployments already limit the types of devices that connect to their network infrastructure—this type of restriction greatly assists QED-type services.

5.7 Wrap Up

Since the initial development and publication of the QED model, there have been several industrial efforts towards automated device quarantine and examination. As a result, a number of related tools and systems have been developed since our initial publication.

These will be reviewed later in the related work section of this dissertation. However, this interest in the QED model only reinforces the value of the basic model.

The development and design of the initial QED prototype had a great deal of influence on the development of our current Panoply infrastructure. QED explored the fundamental configuration model which evolved into the current Panoply implementation, with only a number of small changes. We moved from IPsec to SSL due to simpler configuration requirements, and we moved away from the network beacon-based Panoply descriptors to the voucher-based network discovery and authorization model. The latter model proved to be simpler in practice, as it does not involve extended wireless monitoring in promiscuous mode, and scanning is handled by the wireless device driver.

QED is a prime example of the Spheres of Influence paradigm, following the basic series of configuration steps:

- *Securely discover a relevant group* by identifying compatible and interesting networks, and establishing a quarantined connection.
- *Negotiate for access* through a series of examination and decontamination steps.
- *Connect and interact with the group, subject to the policy of group*, such as limiting local services and maintaining patch levels.

We successfully evaluated QED in terms of basic overhead, and found that in typical use cases, QED imposes minimal overhead. When patches are required, the overhead is acceptable and commensurate with the size of the required update. Additionally, we evaluated QED's effectiveness at slowing the spread of mobile contagion. These results are extremely promising, and definitively illustrate the effectiveness of the QED paradigm.

Chapter 6

Case Study: A Locative Media Application

Locative media applications have the potential to revolutionize the way humans interact with physical locations through the incorporation of virtual representations. Users will interact simultaneously with a physical environment and, with the assistance of a mobile computer, a corresponding virtual representation of that environment. These virtual counterparts will augment physical environments with various forms of annotation—historical information, advertising, recommendations, and even directions. They will also provide interactive entertainment in the form of games, puzzles, music, and even location-sensitive literature. Panoply is well suited to support these classes of applications by providing basic configuration and connectivity services to mobile users, as well as necessary interaction primitives to application developers.

The scenario presented earlier in our introduction to this dissertation describes Bob, a locative media participant. As Bob travels to different locations, his PDA presents a dynamic narrative, where the narrative elements Bob experiences are based upon the actual locations Bob visits, as well as the proximity of others, and also the overall state of the narrative. We have designed and implemented this kind of a group-aware locative media application using the Panoply middleware; additionally, we have worked jointly with a group of students from the English department to develop an interactive, team-oriented and location-sensitive narrative entitled *nan0sphere* that uses our locative media framework. Our narrative is team-based—four participants each take on a role within the story and must act together to completely experience the story. The locative media application and this narrative were deployed on the UCLA campus.

In order to provide a basis for the application and architecture discussion that will follow, we will first describe the narrative. Our locative media application, however, is general purpose and can easily support other locative narratives and even other types of locative media experiences. After our introduction to the narrative, we will present the design and implementation of our locative media application, important issues we addressed in building the application, and our experiences with its use on the UCLA campus. We will also further show the benefit of Spheres of Influence and Panoply by presenting performance measurements of the Panoply *ad hoc* rescue component, which allows the creation and discovery of *ad hoc* content-oriented spheres that assist with media delivery to improve availability and reduce access time.

6.1 The Narrative

Our narrative, *nan0sphere*, was designed to be an interactive, location-and character-driven work of speculative fiction that involved a wide variety of different physical locations on the UCLA campus. *nan0sphere* was co-authored by a UCLA graduate student in the English department and two undergraduates (one in English and one in Computer Science). The goal of this project was to demonstrate cooperative storytelling in the form of a locative media experience with implicit group interactions. *nan0sphere* is experienced in teams of four, in which each member assumes the role of a story character. Each user receives a customized experience, which includes pieces of the story and clues, based on his location, the location of other team members, and the overall progress of the narrative. Each user has the opportunity to manipulate virtual objects and perform virtual actions that affect the narrative state. Individually, no player experiences the entire narrative; only by combining the group members' experiences does the entire narrative emerge.

The story is a speculative, fictional narrative about nanotechnology on the UCLA campus. Four users each play a different character (a security guard, a graduate student, a campus information technology specialist, and a professor of sociology) and interact with the campus from that person’s perspective; each character is received differently in the different locations based upon their job and associated roles. The narrative is goal-driven, and uses in-story, location-based goals as the impetus for each character to move from location to location. Each user carries a mobile device—such as a small laptop, or a handheld PC, with which they interact with the narrative.

The authors used the construction of a new biotechnology building on UCLA’s campus as the main plot device for the narrative. The story begins with the theft of an extremely dangerous prototype technology from a campus nanotechnology lab. The story takes the four users through eight specific points on the campus. When the participants arrive in the relevant locations, descriptions of the location and narrative text appear on their mobile devices. As each character visits more locations, the true story behind the theft is revealed. Each character has a different reason for being involved in the story; for example, the graduate student is in danger of losing her funding, while the sociology professor is best friends with the head of the lab that was burglarized.

The story has a definite plot arc, with each player entering at the "beginning" of his or her involvement in the story. Players gather clues at each location and can engage in virtual conversations with other characters by selecting dialog and action options on their mobile device. When multiple players gather in a single location, new portions of narrative are revealed—in fact, fully exploring the plot requires different combinations of characters to gather in different locations.

In addition to the exploration of the central storyline of *nan0sphere*, the authors wanted to provide a more conceptual media experience within the context of the story. Using the same locations, the authors created three other alternative *paths* that users could take, allowing them to experience the same story from other, and somewhat unexpected, perspectives. These paths use different forms of narrative, ranging from poetry and song to drama and prose, freely quoting other works in order to form complex layers of experience. The *nanite* path follows the stolen swarm of nanotech-based robots as they become self-aware; the *future* path considers the UCLA campus in a post-nanite world; the *Wesley* path follows the thief who originally stole the technology, and traces his descent into madness.

The authors also wanted to create an "infectious" paradigm within the story. When any of the four players come close to being infected with the stolen nanites, they can "jump" to the nanite path for a moment as a way to suggest infection; the same would occur if any of the players accidentally came too close to Wesley: they can choose to enter his path and explore his mind.

With an understanding of our locative narrative in mind, we can now consider how such a locative media experience can be supported within Panoply.

6.2 Supporting Locative Media

In order to maximize the reusability of our locative media application, we designed a modular, tiered architecture to support general locative media experiences. This system consists of three primary components—the underlying Panoply middleware, the Locative Media Application, and the actual narrative content being interpreted by the engine. This basic architecture is depicted in Figure 19; the breakdown of the locative media application will be discussed later in this chapter. Narrative content is dynamically retrieved through Panoply interactions, and interpreted and rendered by the Locative Media Application layer. This modular architecture will allow new content to be dynamically and easily deployed to end users.

A number of important Panoply features simplified the development and design of the locative media application. Each participant in the locative media application carries with them a Panoply-enabled mobile device that represents the user's presence. The local device sphere manages the user's participation in a social sphere which represents the team of participants, and also dynamically discovers nearby locative media participants for content and state sharing, as well as for purposes of narrative adaptation.

This section will discuss the application's design and the Panoply features utilized by the application.

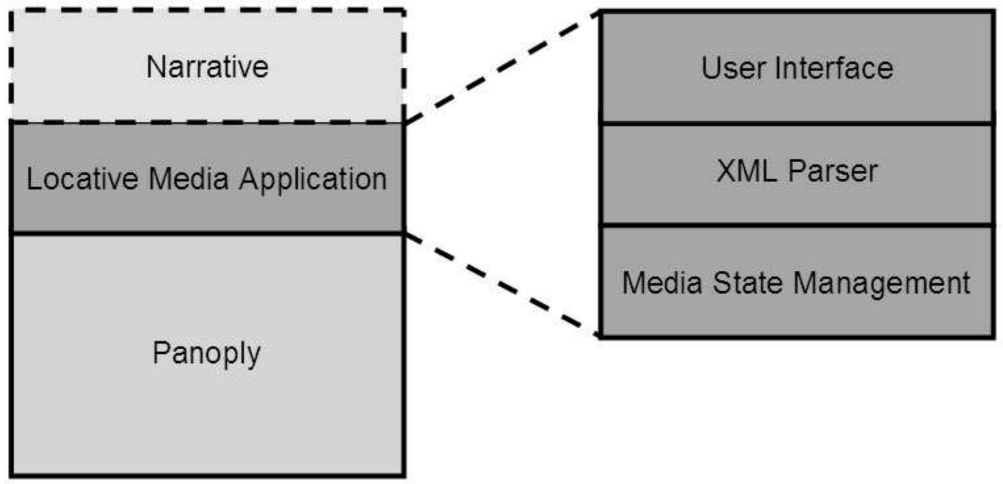


Figure 19. Basic architecture of the Locative Media application

6.2.1 The Locative Media Application

Each user device participating in a locative media experience is running the **Locative Media Application** (abbreviated *LMA* to distinguish it from the larger locative media experience consisting of multiple interacting devices and users). When the user arrives at a location relevant to the narrative, new narrative data is provided to the user's LMA. This narrative data includes both media content as well as state updates describing the overall progress of the media experience. The LMA must properly integrate this information and present it to the user. This functionality is decomposed into three main tasks: *XML parsing*, *media state management,* and *user interface.* Each of these will be discussed in further detail below.

XML Parsing and the Locative Media Markup Language (LMML)

To support dynamic locative media, we have developed an XML-based Locative Media Markup Language (LLML) consisting of basic media elements such as location descriptions, variable management, conditional scenarios, and actions that the characters can take. Additionally, we provide social conditionals that let the media engine test various constraints such as the presence of another team member or the presence of someone exploring a different media piece.

Figure 20 shows a sample location description encoded in LLML. This example describes *Morrison's* Office, a location in the narrative. There is an associated conditional scenario that is activated when the participant is playing the “Rowan” character, and the “Renata” character is also present in the vicinity.

The LMA parses the location description as encoded in the LMML; however, each location description contains conditional scenarios and actions that are only relevant to certain characters or certain combinations of characters, such as in the previous example. To display this content properly, the parsed content must be interpreted in the context of the currently bound set of document variables.

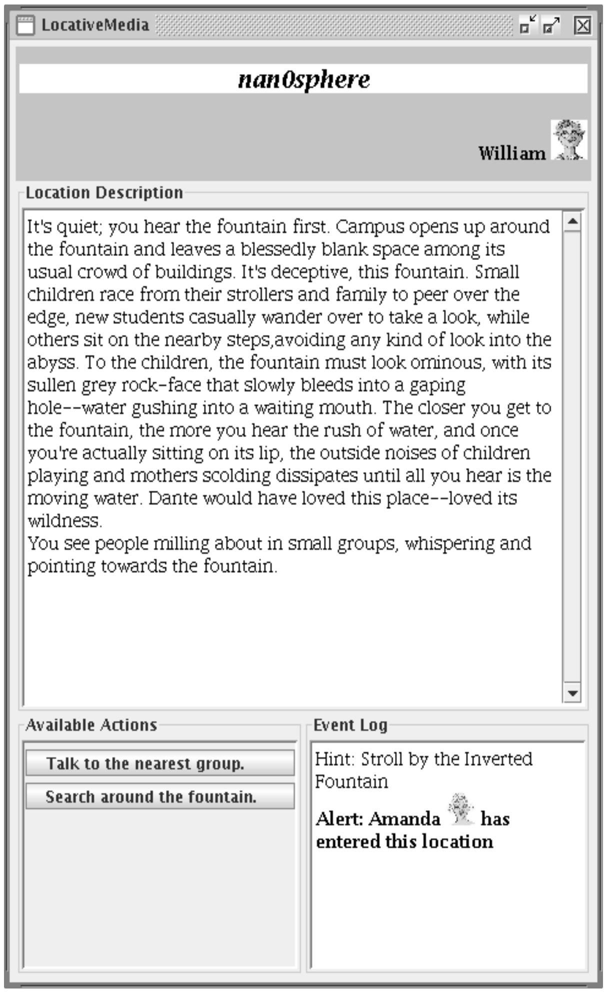
Media State Management

The LMA manages this set of local media variables. When the device sphere receives a media update, this LMML-encoded document contains, in addition to the actual narrative content, the current state of the media experience. This state is encoded as a series of application-level variable bindings. The LMA parses these bindings and stores the variable settings in a local hashtable.

```
<LocationDesc name=”Office” path=”Narrative”>
  <LocationText>Morrison's office had become a second home. Bookshelves
covered one wall. Books, old journals, sheaves of paper, and photos of
family and friends covered the shelves. Mementos from travels to Thailand,
Germany, Australia dotted his walls. A box of blocks and stuffed animals
lay in the back corner of the office, just in case Paulette, his two year-
old visited for the afternoon. He liked to surround himself with things and
images that made him comfortable, especially since he dedicated so much of
his time to research.</LocationText>
  <Scenario>
          <Cond>
                <And>
                       <ActorCondition var="name" val="Rowan"/>
                       <PresentCondition name="Renata"/>
                       <ValueCondition var="Office_Rowan" val="false"/>
                </And>
         </Cond>
         <LocationText>A frazzled student answers the door. "Yes?"<p/>"I
need to talk with Professor Morrison. Right now. It can't
wait!"<p/>"Professor Morrison, I'll talk with you later. Call me if you
need anything." The student opens the door and leaves. </LocationText>
  </Scenario>
</LocationDesc>
```

Figure 20. Sample LMML description for the “Morrison's Office” location

After the location description has been parsed, this state is used to select which scenarios and actions are valid for the current character, given the current state of the media experience. After this is complete, the content is rendered via the LMA's graphical user interface.


LocativeMedia

nan0sphere

William

Location Description

It's quiet; you hear the fountain first. Campus opens up around the fountain and leaves a blessedly blank space among its usual crowd of buildings. It's deceptive, this fountain. Small children race from their strollers and family to peer over the edge, new students casually wander over to take a look, while others sit on the nearby steps,avoiding any kind of look into the abyss. To the children, the fountain must look ominous, with its sullen grey rock-face that slowly bleeds into a gaping hole--water gushing into a waiting mouth. The closer you get to the fountain, the more you hear the rush of water, and once you're actually sitting on its lip, the outside noises of children playing and mothers scolding dissipates until all you hear is the moving water. Dante would have loved this place--loved its wildness.
You see people milling about in small groups, whispering and pointing towards the fountain.

Available Actions

Talk to the nearest group.

Search around the fountain.

Event Log

Hint: Stroll by the Inverted Fountain

Alert: Amanda has entered this location

Figure 21. Locative Media user interface

User Interface

The LMA presents users with a Swing-based graphical user interface (GUI) as shown in Figure 21. The interface presents the user with portions of the narrative; the displayed content typically describes the user's current location, and reflects the progress of the story based on what parts of the narrative have already been explored, and what actions have been performed by the various participants. In the example shown, the user has assumed the role of William, one of the story characters. William has just arrived at the *Inverted Fountain.* The main panel marked "Location Description" displays relevant story text, describing the scene as it would appear at the virtual *Inverted Fountain*. The interface also indicates important events that provide the user with more information and also supplies contextual hints or prompts in an "Event Log" panel. The user can also influence the course of the story through actions; choices for these are provided in an "Available Actions" panel. As the user moves around campus and enters other locations, the interface automatically updates the scene description and the available actions that the character can perform.

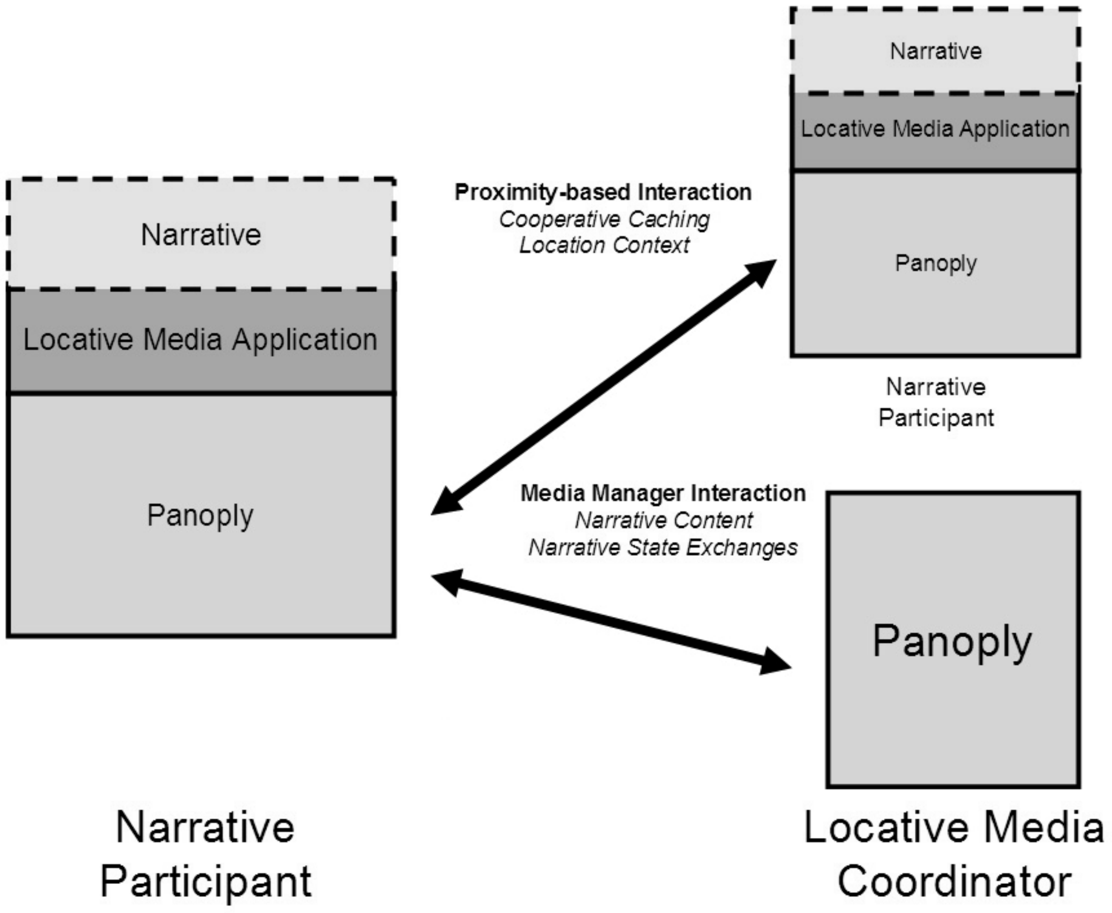

Narrative
Locative Media Application
Panoply
Narrative Participant
Proximity-based Interaction
Cooperative Caching
Location Context
Narrative
Locative Media Application
Panoply
Narrative Participant
Media Manager Interaction
Narrative Content
Narrative State Exchanges
Panoply
Locative Media Coordinator

Figure 22. Interactions within the Locative Media Application

6.2.2 The Locative Media Application as a Panoply Application

Panoply directly supports the LMA in a number of important ways, providing network connectivity, event-based communication, location interfence, and group membership facilities. The Panoply sphere interactions are illustrated in Figure 22. *nan0sphere*, and the locative media application in general, require the creation and management of groups of entities; these include location-based groups, such as all the characters who happen to be in the library at a given time, as well as social groups composed of team members. For our application, each team is represented as a social sphere that manages story state and content dissemination. Each participant's device sphere attempts to maintain a connection to this social sphere. This sphere is hosted by a Panoply server operating in the LASR offices, but it is not required to run an instance of the LMA. Instead, the social sphere's stores the narrative content and state variables for the locative media experience as queryable *SphereState.*

Additionally, peer device spheres can discover one another and form transient location spheres. These spheres provide relationship context, which can be used as an input to the narrative; additionally, the device spheres may potentially assist one another, e.g., by providing cached content to other spheres that are unable to acquire network connectivity.

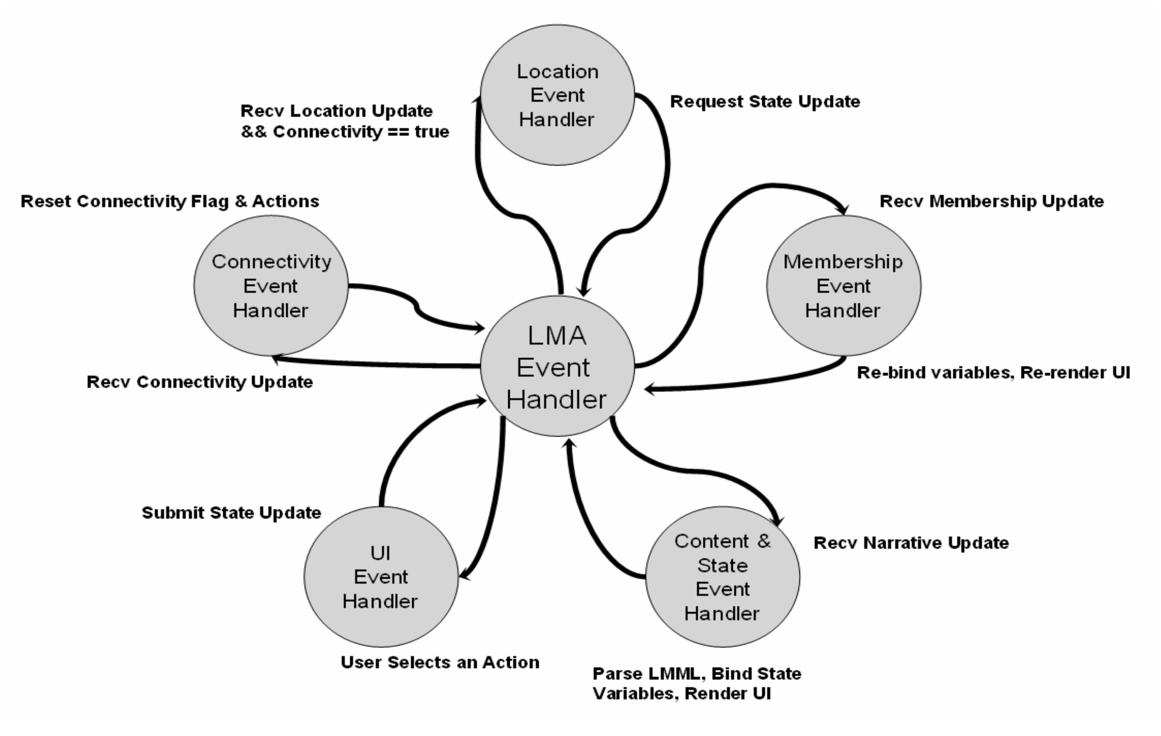


Figure 23. Event handling within the Locative Media Application

The LMA makes extensive use of system-level Panoply events to interact with the local device sphere and also to communicate with remote participants in the locative media application. The LMA registers with Panoply for *CONNECTIVITY, LOCATION,* and *MEMBERSHIP* events. Additionally, the LMA uses directed *STATE* events to query the appropriate social sphere and peers directly for content updates.

The LMA's event handling state machine is depicted in Figure 23. The various event handlers will be discussed starting with the connectivity event handler, and then proceeding clockwise around the diagram.

Connectivity Management

The LMA receives connectivity updates that are generated by the Panoply middleware. These updates provide information regarding the network connectivity status of the local device sphere. Based on these events, the LMA is aware when the sphere is connected to an 802.11 network. This information is used to customize other event handling. Upon losing connectivity, any available actions in the current location are disabled. Actions are disabled to prevent users from selecting an action while disconnected, and possibly entering an inconsistent state with respect to the team view of the shared media.

Location Management

Location updates and refresh events are also delivered to the LMA. When the LMA receives an update informing it that the device sphere has entered a new location, the LMA responds by requesting a narrative update from the locative media social sphere. If the device does not have network connectivity or an active handle to the locative media social sphere, the LMA waits until both of these conditions are satisfied before requesting new content or updating the GUI. Location information is also used to help select a network when the device does not have connectivity. The LMA uses an environment-dependent location network map to assist the device sphere with acquiring network connectivity. The LMA is designed such that location changes do not automatically trigger network discovery if the device already has network connectivity.

Membership Awareness

The LMA also subscribes to and receives membership updates regarding the groups in which the local device sphere is participating, as well as information regarding peer membership in those groups. This information is used to track when the LMA has a live connection to the team social sphere, in order to know when it reasonable to request content and submit state updates. Additionally, these updates are used to alert the LMA to the presence of another team member in the same location. Team members identify one another and dynamically create application-oriented spheres. This is done automatically by Panoply's interest manager, which constantly monitors the environment for the presence of other team members. Within these spheres, they can share information regarding their respective locations—this provides social-location context that is used to determine who is in the immediate physical area for narrative purposes.

Narrative & UI Event Handling

Narrative content and narrative variables are exchanged in Panoply state events. These are typically exchanged between the device spheres and the locative media social sphere and are encapsulated inside *directed events* so they are directed to the proper destination, and only that destination. The parsing and rendering of the content is handled as described earlier in Section 6.2.1.

6.3 Lessons and Experiences from nan0spheres

Our experience with the deployment has yielded interesting insights into design issues for locative media and locative media infrastructure, as well as issues and questions for authors developing location-aware media. The relationship between software author and storyteller is significantly blurred as infrastructural limitations feed into the narrative, and narratives approach the level of software in their complexity.

Social Issues in *nan0sphere*

It became clear that the narrative's dependencies on coordinated actions taken by multiple characters could be problematic. The story could only progress when different characters took specific actions, pushing the story's progress forward. At a given moment, any given character might find that he had no options in any story location because the story was blocked, waiting for some other character to make progress. If a participant took a break from the story, he could effectively prevent other characters from accessing new content and completing the narrative. Depending on author intent, this might be not acceptable. One possible solution to this problem would be to allow optional narrative event timers that would ensure that the narrative advances at a reasonable rate by triggering unresolved game events necessary for active characters to progress. These should be optional, because a content author might, in fact, wish to require that a specific plot event occur in a specific manner.

Another difficult challenge that we encountered with *nan0sphere* was the question of how to engage the participant and motivate them to want to walk around campus. Keeping users' attention is simpler when locative media is experienced in small and constrained space; when the environment is a large campus, how can an author sufficiently capture the users' interest and inspire them to trek multiple times around a mile-wide campus visiting and re-visiting locations? *nan0sphere*'s narrative "bounced" between physically distant locations on the UCLA campus. To fully explore the story, participants traveled back and forth between different story locations. Sometimes a character would arrive at a new location, only to be told to go back to the last location she visited. Unless the point of a narrative is to encourage exercise, forcing user mobility can be tedious. If a narrative is to convince a participant to change locations, the narrative must offer sufficient allure to overcome human inertia. A compelling narrative, or some form of competition and reward may be sufficient.

The decision was made early to make *nan0sphere* a plot-driven mystery and use clues and cliffhangers that propel the narrative forward and encourage people to walk around the campus to try and find more clues. Experience with running users through the story suggested that this approach was not sufficient for the amount of user movement the story demanded. Perhaps a narrative designed to serve as a tour of an interesting area could solve this problem. One promising possibility to avoid too much user movement is to restructure the notion of locations in the locative media. Events in locative media may not be tied to a specific location, e.g., "Café Roma," but rather to a type of location, e.g. a café or restaurant, etc., or location *template*. Using this technique, a locative narrative might progress as the participant goes about their daily routine, only forcing particular movements for major story events.

The project's conceptual layers (the different "paths" one can take to reveal more of the story) were a response to the artistic constraints that such a plot-driven story implied. They enabled the authors to experiment with prose styles and use the software in novel ways with regard to location and user interaction. The authors needed to agree on what kind of story they should construct and who their intended audience was. They vacillated between wanting to present their audience with a very abstract media experience that worked with the same concepts (nanotechnology, the body, the relationship between scholar and subject), but relied on users to draw their own connections as to how they would navigate through the project, and a straightforward narrative that presented a "real" story, one with reasons behind every action. Here is the fundamental divide that was encountered when creating *nan0sphere*: what constitutes a *real* and maybe more importantly *enjoyable* story? It proved difficult to create a narrative that was exciting conceptually, yet concrete enough so that users would feel they were really getting somewhere.

Debugging Interactive Narratives

While refining our framework, we developed a number of debugging tools. We found that it was desirable to exercise the application without actually moving about the campus, and thus we created a clickable map of the campus to simulate location transitions. Additionally, we built a version of our media interpreter that displayed and logged the conditional decisions affecting the current story. In our experience, it would be useful to have a comprehensive debugging framework so that developers and authors could easily isolate narrative components and test them under both real and simulated conditions.

For example, in the narrative description language, the authors were able to specify what text they wanted to associate with various locations. They were also able to specify various conditions that controlled when certain portions of the text were made available for display. During testing, it became evident that the authors did not, and with available tools could not, completely anticipate all possible paths that individual characters could take. On occasion an individual user's experience might include character introduction or plot development that seems to be "out of order," at least to the extent that text in interactive media can be so. Clearly, we need better support for authors to express high-level flow constraints on their stories, akin to software invariants. One role of a debugging framework could be constraint verification on the narrative content; this would point out possible narrative flow problems to the author.

Localization Issues

Typical devices participating in location-based interactive narrative applications need to retrieve dynamic content relevant to the current location; social applications also require maintained relationships to peers and story groups. This mandates that devices must acquire and maintain network connectivity.

As users explore the campus with their mobile device, Panoply monitors the wireless landscape to identify appropriate 802.11 networks, and attempts to maintain network connectivity and a connection to the social sphere representing the character's team.

Network selection is based on detected location. This application was developed before we had completed our localization map provisioning service, and thus this application uses a static localization map provided at installation time. Extending this application to use dynamically loaded maps would be a simple exercise.

The *nan0sphere* authors selected eight locations on campus to be semantic regions that play a part in the narrative. Some of the referenced locations are highly specific, intending that the user be in one small area, such as an individual room, or a particular bench in a garden. Others are intended to be more broadly defined and aim to have the user in the general proximity of a landmark. In both cases, accurate localization is important. In the former case, we wanted to be somewhat lenient regarding how precisely locations were defined. In practice, users were discouraged if they went to an area specified in the story, but were unable to situate themselves in the exact position the authors envisioned and thus failed to localize. By defining a slightly larger zone, users need only approach the general area to know that they are on the right track. Three of the locations chosen are particularly close to one another. Two of these are outdoor locations and one is indoors. Although these regions do not overlap, they are close enough that it can be difficult to differentiate one from another with high accuracy, clearly presenting difficulties for participants. This is a limitation of our localization technique; however, in general, some limitations will exist in many localization schemes. Accordingly, content authors need to be aware of the limitations of the localization support and design.

When the device determines that it has moved to a new location, our prototype gives both auditory and visual cues. The auditory cue was added during debugging, and though it is configurable, we have typically left it enabled. We discovered that users tend to focus their attention on their devices when they changed location, possibly as a result of the tone. The change in location results in a corresponding change of text displayed to the user. Users tended to immediately read the new text and proceed with the story directly from that location. Thus, in the cases where the user was supposed to reach a specific point, they sometimes did not get to the authors' exact intended location before progressing with the story. It may be possible to modify the interface to inform users when they are getting “warmer” so as to lead them all the way to the intended location before allowing the story to progress.

6.4 Rendezvous Rescue in the Locative Media Application

While 802.11-based connectivity is theoretically available in all of the areas relevant to *nan0sphere,* in practice, Internet connectivity is not always available. There were a number of places of interest where there were substantial gaps in 802.11 connectivity or where low signal strength prevented successful association—at times it took wandering around an area for several minutes before acquiring connectivity. This problem led us to develop and add an *ad hoc rescue* facility to Panoply. When no infrastructure-based 802.11 connectivity is available, the device sphere seeks to form an ad hoc sphere for the purpose of acquiring a narrative update.

Panoply rendezvous rescue uses Panoply’s network primitives and the Spheres of Influence discovery and group formation facilities to dynamically create a transient 802.11 ad hoc network and form interest- or task-based spheres. This functionality is designed to allow the device sphere to seek out available *rescuers* who can assist by providing application content or configuration hints, or even by acting as a type of network proxy when the *rescuee* cannot identify or configure itself for an appropriate infrastructure network.

There are clear security implications to rendezvous rescue, not just for *nan0spheres,* but also for general applications and services that might wish to make use of this functionality. Clearly, rescuing spheres should possess policy that requires the presentation of appropriate credentials before allowing access to local services or privileged information. Additionally, the potential rescuer may wish to determine more of the rescuee's intent before initiating rendezvous rescuer. This and other related issues are discussed more in Chapter 9, Future Work.

6.4.1 The Rendezvous Rescue Mechanism

The Rendezvous Rescue mechanism is implemented as an extension to the connectivity management subsystem of Panoply. The mechanism is split into two parts—the *rescuee* subsystem, and the *rescuer* subsystem.

The rescuee subsystem is controlled with a RESCUE request event which sets a flag indicating that the local sphere is in need of assistance. The rescue request flag is time-limited in order to ensure the relevancy of the request. The next connectivity update indicating that the sphere lacks connectivity triggers an *ad hoc rendezvous mode* in which the device creates a new 802.11 ad hoc cell with a Panoply-specific identifier, and assigns to the wireless network interface an IPv4 link-local (IPv4LL) address [RFC3927.] Thus configured, the sphere behaves as normal—discovering and forming sphere relationships when possible. The sphere remains in the rescue mode during a fixed *rescue window*—since rescue is not a sure thing, the sphere does not wish to rely on rescue alone. The sphere will, if it does not get any rescuers, soon reattempt to acquire infrastructure-based connectivity.

The rescuer subsystem monitors 802.11 discovery events for ad hoc ESSIDs that reference the Panoply rendezvous identifier. The current implementation uses a fixed identifier encoded as the 802.11 ESSID (*"prendezvous")*, but this could be extended with Panoply task or application-specific UIDs encoded in the ad hoc ESSID. When a potential rescuer detects another sphere in need of rescue, the rescuer first determines if it is able to offer rescue services by querying local applications. If a local application is in the middle of an operation, it may desire to delay rescue operations. In general, this requires applications to be rescue aware, or to have some knowledge of application protocols. More work is needed here to understand the larger ramifications and feasibility of general device rescue given unaware legacy applications. A second network interface that could be used for rescue purposes would be of great benefit, allowing the rescuing device to maintain an infrastructure connection while performing rescue activities.

If local applications are in an acceptable state to allow rescue, the sphere configures itself to join the ad hoc cell and assigns its wireless network interface a link-local address. This places the spheres within the same network, and Panoply sphere discovery and advertising components handle the formation of any needed interest-based spheres. The locative media application advertises its application identifier, and peers form transient location spheres, as described in Section 6.2.2.

Incorporating rescue functionality into the LMA required two minor changes to the application. The LMA was modified to issue a rescue request when the device sphere lacked network connectivity, and the application also received a non-void location update. The second modification to the LMA allowed the LMA to request location content directly from connected peers when the device sphere lacked an infrastructure network. Network connectivity context passed through connectivity update events provided the LMA with the necessary knowledge to know when it should issue direct queries to peers and when it should wait for the locative media social sphere.

6.4.2 Evaluating Rendezvous Rescue

To evaluate our implementation of rendezvous rescue, we needed a way to test the locative media application with, and without, the rendezvous rescue mechanism. Additionally, this evaluation needed to be consistent and repeatable, in order to gather sufficient data for evaluation. For evaluation purposes, we simulated sphere failure to acquire network connectivity—this captures effects such as the lack of an acceptable network, failing to associate due to weak signal strength, failing to acquire a DHCP address, or failure to set up a successful VPN tunnel. All of these problems were observed in our deployment of the locative media application.

Experimental Methodology

Two device spheres were used for the evaluation—one was the *rescuee,* and the other was a dedicated *rescuer.* Each device sphere was hosted by an IBM T42 laptop, running Ubuntu Feisty. Each laptop was equipped with 512 MB of RAM, and an Intel Pro Wireless 802.11 interface.

In a real deployment, the utility of rescue would be partially dependent on the probability that a rescuer was in near enough proximity to overhear the rescue beacon. For the evaluation of the rendezvous rescue mechanism itself, we assumed that this was always the case. Despite having a dedicated rescuer, rescue is not a sure thing. A successful rescue attempt depends on a number of factors, including timely detection of the rescue beacon and formation of the rescue sphere during the rescue window.

Both spheres ran Panoply installations and the Locative Media Applications. Network connectivity failures were generated by selectively ignoring the results of infrastructure-based network connectivity attempts according to a probabilistic threshold. We tested both, with rescue enabled and with rescue disabled, for network failure rates ranging from 0% to 90% in 10% increments. For the purposes of this experiment, the localization module was statically configured—after the initialization of the localization module, the localization module always returned a fixed location. This was done to remove transient localization effects that would introduce inconsistencies into the experiment. At each failure rate, we measured the time to acquire content within the locative media application 100 times. The time measured included localization initialization, network configuration, IP address acquisition (via DHCP or link-local), and sphere join/policy negotiation overhead, in addition to the data overhead inherent in transferring the content.

Experimental Results

The experimental results are summarized in Table 9 and Table 10 below. Table 11 shows the difference in content access time without rescue and with rescue.

Infrastructure Failure Rate	0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
Mean Time	15757	17237	19176	24214	38806	40653	72858	86985	175144	339339
Std Err	309	507	804	1798	5476	6619	13833	14433	23390	32922
Median Time	14630	15719	15202	16780	19371	27300	32143	32791	46294	228818
75th Percentile	18511	18758	19635	30826	30967	34873	42920	47658	281614	522685
25th Percentile	13705	14324	14136	14556	15188	14866	18027	19058	22749	37229

Table 9. Summarized content access time without rescue support
(all times are in milliseconds)

Infrastructure Failure Rate	0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
Mean Time	15833	18024	18973	22036	25386	29497	31602	31021	38452	43674
Std Err	305	811	762	1123	1463	1839	1943	1813	3561	2506
Median Time	14998	15031	16579	17301	19476	24081	26366	26703	28779	33923
75th Percentile	18600	19106	19739	27383	28201	32644	35114	32841	38412	55599
25th Percentile	13820	13925	14718	14511	15810	16146	18703	19432	24898	26532

Table 10. Summarized content access time with rescue support enabled
(all times are in milliseconds)

Infrastructure Failure Rate	0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
Mean Difference	-76	-788	203	2178	13419	11685	41255	55964	136692	295665
Median Difference	-369	688	-1377	-521	-106	3220	5777	6088	17515	194895

Table 11. Summarized differences with and without rescue
(all times are in milliseconds)

Benefit of Rescue in the Locative Media Application

The above data clearly shows the benefit of rendezvous rescue. In general, when there is a connectivity failure, the spheres are able to enter rescue mode and exchange the location content. Table 11 indicates that when rescue is not necessary, the rescue mechanism imposes a negligible amount of overhead that would likely go unnoticed by the user.

Without rescue support, at probabilities greater than 50% that any given attempt to join an infrastructure network fails, content access time increases quickly into the range of minutes for a portion of the population; this is clearly illustrated in Figure 24. In general, this is unacceptable behavior. A user is unlikely to be willing to wait minutes waving their mobile device around until something happens; our experience with the Locative

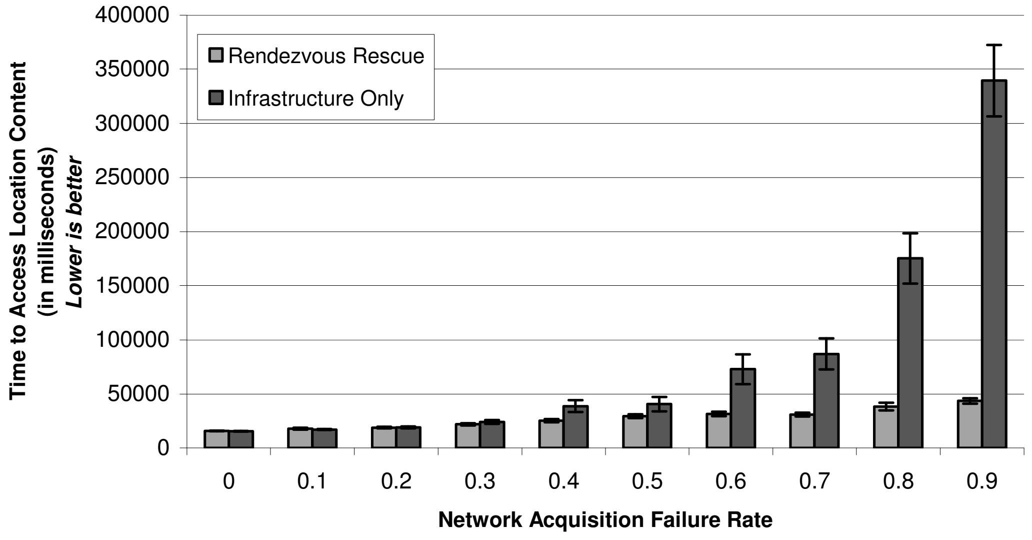


Figure 24. Mean time to access content with infrastructure network failures (error bars indicate standard error)

Media Application indicates that this quickly becomes frustrating to the users.

Rendezvous rescue performs much better; at a 90% network infrastructure failure rate, ~80% of users receive their location content in less than a minute, compared to the ~29% that receive location content in less than a minute without rescue. Additionally, median

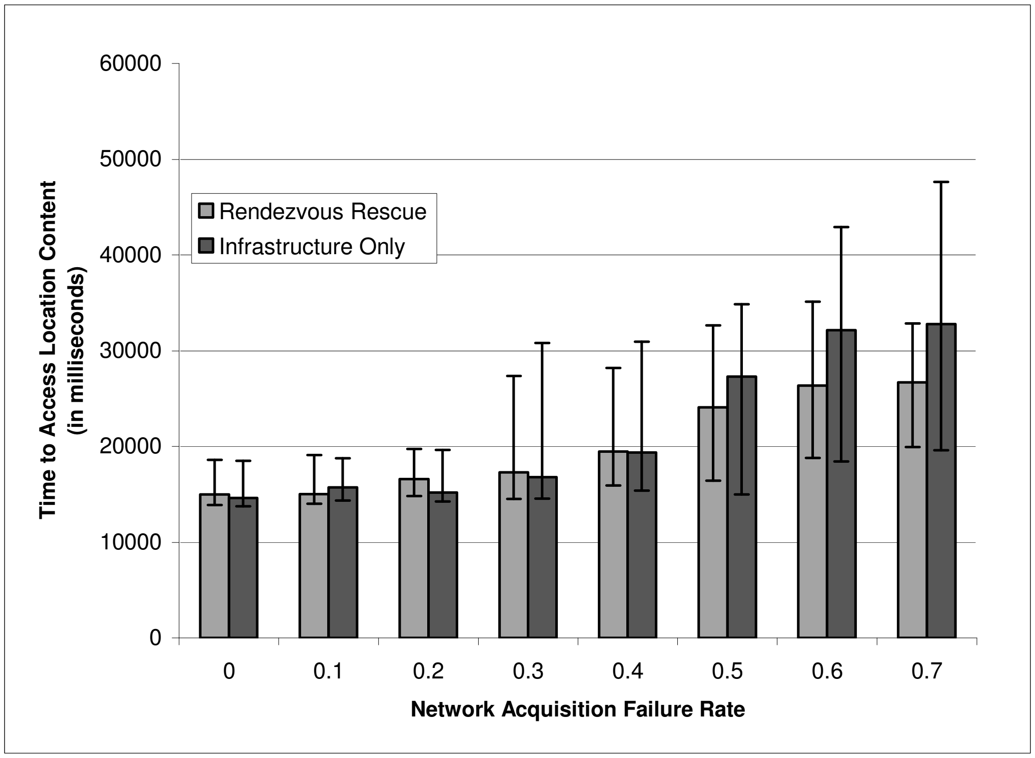


Figure 25. Median time to access content with infrastructure network failures
(error bars indicate 25th and 75th percentiles)

access time for Rendezvous Rescue is < 34 seconds across the board—this is indicative of the typical user experience. At 100% infrastructure failure, content retrieval is only possible through the rescue mechanism, again illustrating the benefit of the paradigm.

These worst-case scenarios make it easy to show the benefit of Rendezvous Rescue. Figure 25 illustrates the median content access times. At low values of infrastructure failure, rescue is nearly indistinguishable from the infrastructure only, no-rescue case. This points out the low overhead of the rescue mechanism—it does not greatly affect normal behavior or content access. As failure rates push past 30%, we start to see a noticeable improvement for the 75th percentile, as expected. This trend continues, with rendezvous rescue providing increased benefits as the probability of network infrastructure failures increases.

6.5 Wrap Up

The success of *nan0sphere* as a locative media application was mixed. We learned much about the realities of building this kind of locative media application, and it helped improve the Panoply infrastructure. Some of tools built in conjunction with *nan0sphere* may be helpful in building other such applications, and the lessons that we learned will benefit other groups. On the other hand, *nan0sphere* did not become popular. Even members of our own group found working through the entire story somewhat tedious, and there was no enthusiasm for running through multiple story outcomes or exercising optional features—in large part due to the sheer amount of physical movement required. Perhaps the single greatest lesson that this application offers is that these types of stories need to consider user mobility carefully. Stories need to provide strong motivations for the activity they require. A story must be extremely compelling to get its readers to walk up and down hills, go into and out of several buildings, and figure out exactly which locations need to be visited next. Alternately, the story should be able to adapt to the actual mobility of the user. In this operating mode, content must be written and presented in such a way that it can be bound in a just-in-time manner to a specific location.

An important lesson in regard to the group aspects of *nan0sphere* is that the group experience must be designed to involve the group, yet not require too stringently that all group members participate at once or experience the story at the same speed. This offers extra challenges in designing such stories.

As a Panoply application, the Locative Media Application was a success—it demonstrated dynamic discovery and the use of ephemeral groups as context to customize a media experience. Additionally, our experiences yielded insight that lead to the development of the rendezvous rescue facility, a valuable Panoply component. The rendezvous rescue facility can easily be used by other applications to provide cached content, exchange networking configuration hints, or even proxy web content to nearby devices in need of assistance.

Chapter 7

Case Study: The Smart Party

Devices such as the Apple iPod® and the Microsoft Zune®, as well as online services such as YouTube, have demonstrated the phenomena of social sharing of media content, both online and in person [Kahney03]. Social content sharing is still in its infancy, but we foresee many possible directions for the evolution of this technology. Ubiquitous computing, with its human- and locale-centric emphasis, shows great potential for enhancing social content sharing applications. The Microsoft Zune, with its music sharing capabilities, is an exciting step in this direction, allowing users to share music wirelessly and easily with other nearby Zune users.

Ubiquitous computing technologies will allow us to greatly extend and generalize upon this simple application. In our example scenario, briefly laid out in Section 1.1.1, we suggest that groups of users will be able to share media content and express their collective user preferences within their shared environments. We believe that this type of application follows naturally from current trends in social media sharing within the context of ubiquitous computing. Groups of users will enter into new environments, and will want to interact effortlessly with each other within their new common environment. Location-specific services will provide a shared experience, customized to the local group of users, based upon their collective preferences and the media contained within their devices.

We refer to this location-oriented variant of social content sharing and preference coordination as the *Smart Party*. Within a Smart Party, user data and context—preferences, media, metadata, location, social memberships, application interests, etc.—could be used collectively and collaboratively to select audio and video, alter environmental settings, coordinate power management, and identify participants interested in an activity, among other things.

The Smart Party is an excellent application to demonstrate the Panoply middleware fully. First, it demonstrates voucher-based network configuration on behalf of a group of users who may have never previously visited the application site. Second, it features dynamic provisioning of localization data, demonstrating the extensibility of the Panoply localization model. In addition, the Smart Party illustrates location-based event scoping, the dynamic formation of interest-based spheres representing common musical interests, and active management of user preferences. This effectively exercises many of Panoply's key features, and clearly demonstrates the value of our approach.

This section examines the requirements of the Smart Party and presents our design and implementation of a Panoply-based Smart Party. We have studied how interest-based peer organization and coordination in the Smart Party can improve system performance and the user experience, and we will also present those results.

7.1 Smart Party Overview

In our vision of the Smart Party, a group of people attend a gathering hosted at someone's home. Each person carries a small mobile device—most likely an enhanced mobile phone or ultra-portable PC, such as an OQO®, a Sony uPC, or the recently announced

Microsoft Origami. The party environment consists of a series of rooms, each equipped with speakers. The home is covered by one or more wireless access points that have been secured; additionally, neighboring homes may possess access points that are also visible from the party space. This starting setup is depicted in Figure 26, which depicts our group of guests who have just arrived at the Smart Party environment.

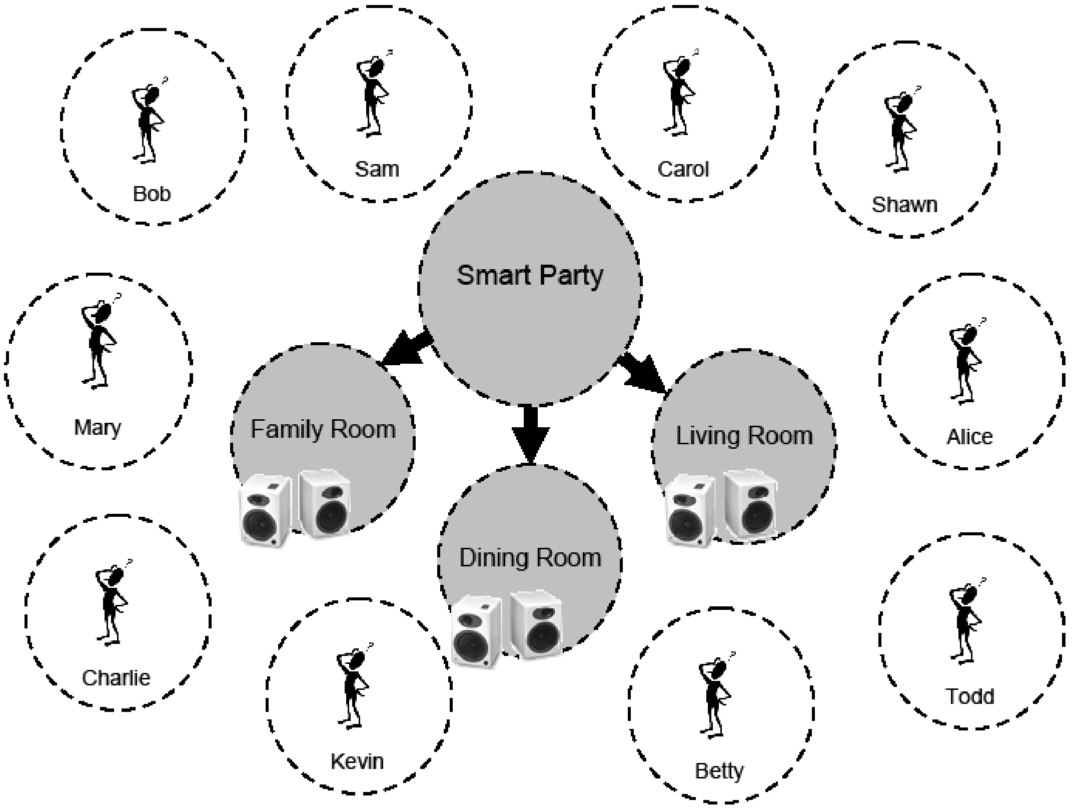


Figure 26. Before the Smart Party
Guests are not yet connected, and have no concept of the Smart Party, its network, locales, or other guests

As each invited guest arrives at the party environment, his or her mobile device securely and automatically associates with the correct network. As the partygoers move into the rooms of the party space, each room programs an audio playlist based on the current room occupants, their communal music preferences, and the content they have brought to the party. Guests automatically and dynamically collaborate with one another to form preference-based coalitions that manage their collective preferences and steer the music choices for the Smart Party. One vision of a fully formed and configured Smart Party is shown in Figure 27. In this depiction, the party is represented by a master sphere that coordinates communication and guest configuration. Under that sphere, each room is represented as its own location sphere which manages the audio hardware and playback, and coordinates guests beneath it hierarchically. Under each location sphere, guests have formed into a number of genre-specific interest spheres that coordinate their voting preferences to achieve a communal goal—e.g., hearing a certain genre of music. In the living room and dining room, multiple genre-based coalitions compete with each other for members; both contribute to the audio preferences of the environment.

Our earlier discussion of the Smart Party in Section 1.1.1 raised a number of technical issues that Panoply solves, including appropriate network selection and configuration, guest authentication and authorization, and acquisition of localization data, as well as basic guest coordination facilities. We will now discuss the technical requirements of the Panoply-based Smart Party, and explore how Panoply solves the configuration and management challenges for guest devices at the Smart Party.

The Smart Party is a location-aware application in which guest devices arrive at an environment, interact within a shared semantic location, e.g., a room in the party, and dynamically select and provide media that will be played in that location.

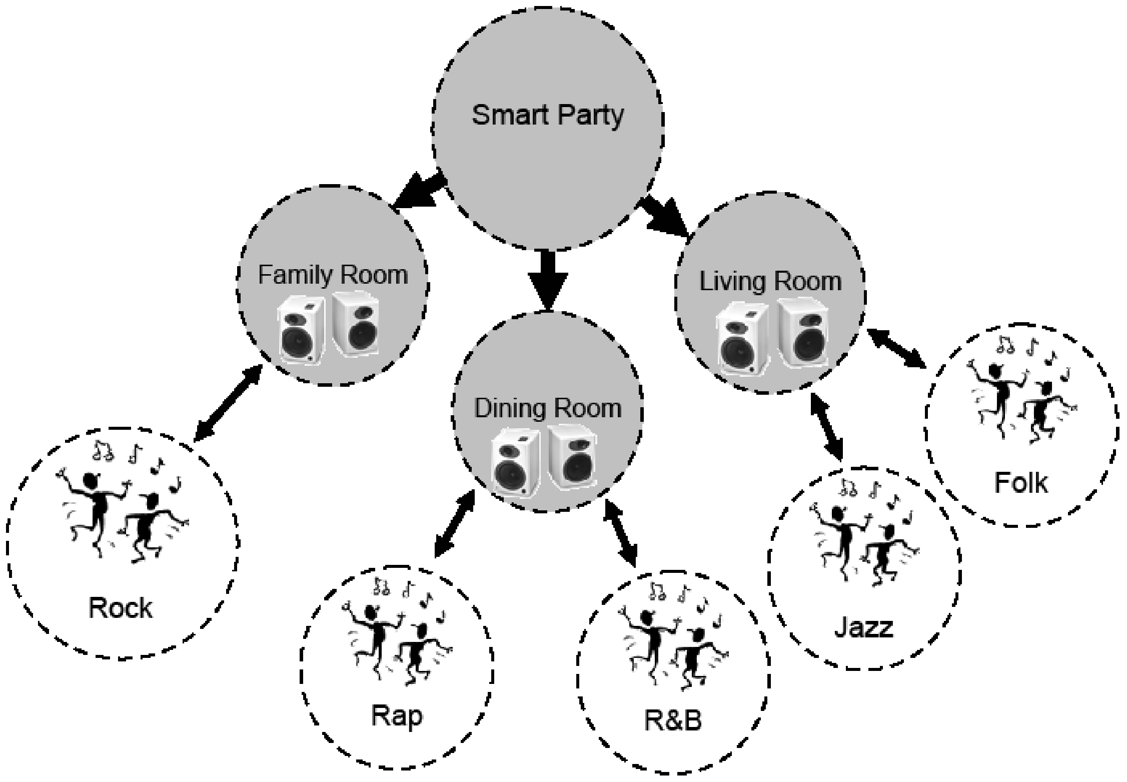


Figure 27. A configured Smart Party
Users are organized into interest-based groups and collaborate to
select and provide media within the Smart Party

At a high-level, this application consists of a number of functional components, many of which are handled automatically by Panoply. Breaking down the requirements of the Smart Party, we identified the following necessary functions:

- Configuring and authorizing guest devices
- Localizing guest devices
- Media suggestion and search

Each of the functions will be broken down into a detailed description of the functionality, services provided by Panoply, and new functionality that was required by the Smart Party application.

7.2 Configuring and Authorizing Guest Devices

The challenge concerning configuration and authorization is that of efficiently setting up transient associations between the appropriate guest devices and the emergent party device community. For simplicity, let us refer to the host of the party as John. In the existing wireless world, if John wanted to allow a group of guests, such as those shown in Figure 26, to use his secure 802.11 network, current best practices would involve the following process:

John would need to give his guests the 802.11 network identifier (an SSID), and also a pre-shared key (e.g., for WPA-PSK) or an authorizing certificate to provision EAP-TLS. At worst, this requires John to interact with each guest's machine as they arrive at the party, and at best, requires each guest to interact manually with a side-band authentication and configuration provisioning system, e.g., a USB or infrared-based introduction system such as Balfanz et al.'s Network-in-a-Box [Balfanz2004]. At this point, authorized users would have full access to John's network.

This process requires John to take actions at his house when guests arrive, on behalf of every guest. This is far from ideal. Additionally, different locations may require different side-band techniques, and not all guest devices may possess the necessary side-band channel.

The Smart Party, like other Panoply-based applications, leverages the Panoply voucher and configuration and authorization primitives to enable invited guests to automatically detect, associate, and securely authorize themselves to the party environment. In the Smart Party application, John issues vouchers to all of the intended guests (cf. Figure 28). The invitation vouchers are typically disseminated through email. Invitation vouchers could also be disseminated via an invitation management service similar to evite.com. Alternately, the invitation vouchers can be disseminated through existing sphere connections, or through a USB-based location-limited sideband protocol at the Smart Party location, though this latter mode has physical scaling issues.

These vouchers encode a description of John's network environment (one or more MAC addresses), and a cryptographic invitation specific to the guest that includes network configuration information such as a network SSID, a validity period, and pre-shared keys or TLS certificates as necessary. Additionally, the voucher specifies which application to execute when the voucher is activated—in this case, the SmartPartyUserDeviceApplication.

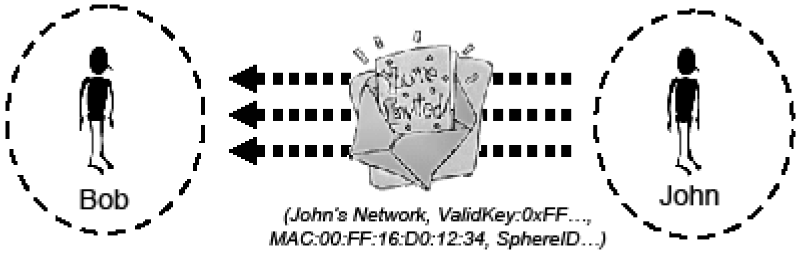

Bob
John
(John's Network, ValidKey:0xFF…,
MAC:00:FF:16:D0:12:34, SphereID…)

Figure 28. Secure voucher-based invitation is transmitted to party invitee

Bob, one of John's invited guests, has received an invitation voucher, and has come to John's home to attend the party. When Bob's device—a PDA, for instance—enters detection range of John's network, the WiFiDiscovery module running in the device sphere delivers events describing the 802.11 environment in terms of visible access points and their respective signal strengths.

The Panoply voucher manager checks these observations against its voucher database, and determines that the Bob's voucher for the Smart Party matches the external observations. Since Bob has arrived within the timeframe in which the voucher is valid, the voucher is activated. The voucher manager submits configuration and join events into the device's event processor. Bob's device associates with the specified network using the security parameters specified in the voucher, and attempts to join the specified sphere. Bob's PDA then negotiates with the local sphere for access to the Smart Party. The device provides its voucher (cf. Figure 29) as evidence of its authority to join the sphere, and is granted access to the local sphere. Bob's PDA then launches the specified application, the *SmartPartyUserDeviceApplication* (SPUDA). All of this functionality was provided entirely by the Panoply middleware. At this point, Bob's device is configured and authorized to interact with the local sphere, but has no notion of its location within John's home and cannot yet fully participate in the Smart Party.

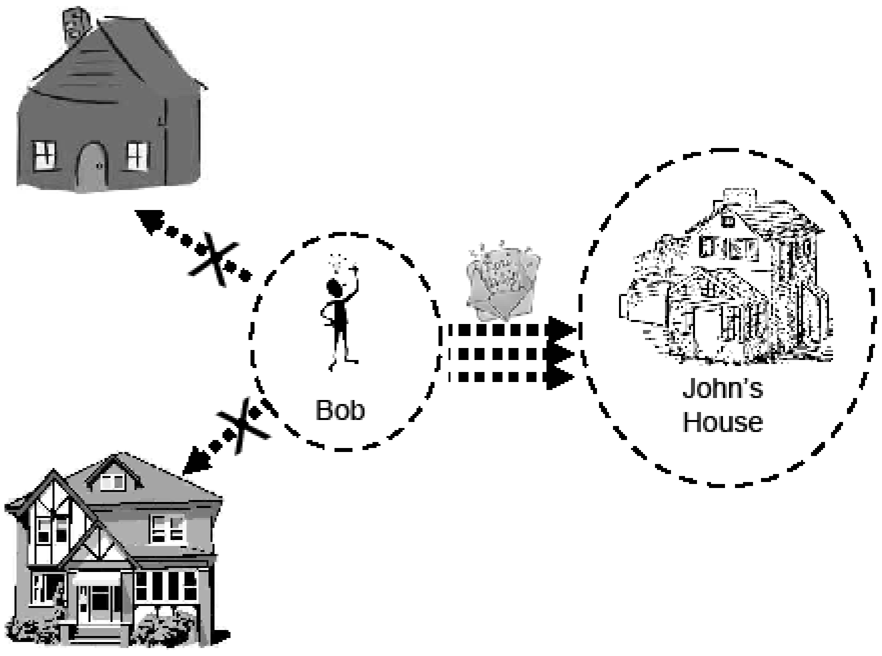


Figure 29. Guest voucher is used to select a network, and authorize participation

7.3 Localizing Guest Devices

The second function of the Smart Party application is enabling guest devices to localize themselves properly within the party environment. This is necessary in order for the guest devices to identify the appropriate room-level sphere they should be participating in at any given time. Panoply already provides all of the necessary support mechanisms to allow guest devices to extend their localization facilities, allowing them to operate within their new environment.

In the Panoply paradigm, after the guest devices have discovered the Smart Party environment and associated and connected to John's home sphere, they must be given localization maps to enable them to determine their position within the party environment. Let us return to Bob, who is in need of location configuration. Bob's PDA, like every device running Panoply, possesses a localization map management service within its device sphere. Bob's PDA is running the *SmartPartyUserDeviceApplication*, which triggers a localization map request; a corresponding *LocalizationMapRequestEvent* is delivered to the Smart Party sphere. The sphere responds with a *LocalizationMapUpdateEvent* containing the necessary 802.11-based fingerprints to enable the guest devices to localize within the Smart Party. The location Manager on Bob's PDA processes the new localization database, and begins using the database. When the guest device localizes, the location manager provides location updates that describe the localized region with its semantic identifier, e.g., **John's Living Room** and also with the ID of the *room sphere* associated with the semantic region. This enables the device sphere to connect and join with the room sphere.

7.4 Media Suggestion and Search

After localizing and joining the appropriate room sphere, the guest devices are ready to participate fully in the Smart Party. In this phase of operation, guest devices must interact with the room sphere to select and provide media content to the room sphere. Each room sphere is running a Panoply application—the *SmartPartyLocationApp*. This application is responsible for managing the room's media operations and the content provisioning protocol. The content provisioning protocol (CPP) is separated into three phases: *suggestion, search,* and *delivery.* This functionality is implemented by the Smart Party applications, using Panoply event and scoping primitives.

When members are detected within a room, i.e., the member count of the location sphere is non-zero, the corresponding SmartPartyLocationApp, also known as the SPLA, initiates the suggestion phase of the protocol by sending a location-scoped *MediaSuggestionRequestEvent*. This event, delivered only to interested recipients within the scope of their shared location, initiates the CPP state machine in the SPUDA running on each guest device. The SPUDA consults the current user preferences to identify a piece of content to suggest to the party. Once the SPUDA has determined which piece of content to suggest, it returns a *MediaSuggestionResponseEvent* scoped to the local room, which is picked up by the room sphere and the SPLA.

Given a set of incoming media suggestions, the SPLA selects a piece of content and then initiates the content search phase of the CPP by generating a *MediaSearchEvent*. In the search phase, the SPLA seeks to identify a peer that possesses the specified media and is willing to provide it to the party. When guest devices receive the *MediaSearchEvent,* and they possess the specified content and are willing to provide it to the SPLA, they return a *MediaSearchResponseEvent* to the SPLA. If no search response is returned in a timely fashion, the SPLA starts the process over by returning to the suggestion phase of the protocol.

Once the SPLA has received a *MediaSearchResponseEvent,* it creates a *DirectedEvent* addressed to the responding sphere, and attaches a *MediaDeliveryRequestEvent*. The SPLA and the content-providing sphere use directed events to exchange the media content. Once the full media content has been received, it is added to the playlist managed by the SPLA and played in turn.

Let us return to our example to demonstrate the user experience of this phase of the Smart Party. Bob has entered the living room at John's house, and his device has just localized and is establishing a connection to the living room location sphere. As Bob is the first guest to visit that particular area, music is not yet playing when Bob enters. Once the living room sphere detects Bob's presence via the membership event generated by Bob's device sphere joining the room sphere, the living room sphere asks Bob's PDA for music suggestions. Bob's PDA quickly examines the available local content and preferences, and determines that Bob would like to hear *Bob Dylan – Blowin' in the Wind*, and suggests that song to the living room sphere. Given no competition, the living room sphere agrees to the suggestion and issues a search request for the song within the context of the Smart Party. Bob's PDA responds back that it has the song, and the living room initiates a transfer of the song to the audio playback hardware. Once the song is fully transferred, the location application starts the media player, and the Bob Dylan song fills the room.

7.5 Smart Party Configuration and Implementation

Now that we have walked through the mechanical procedure through which guests discover, join, and configure themselves to interact within the Smart Party, we can discuss the overall Smart Party configuration and implementation.

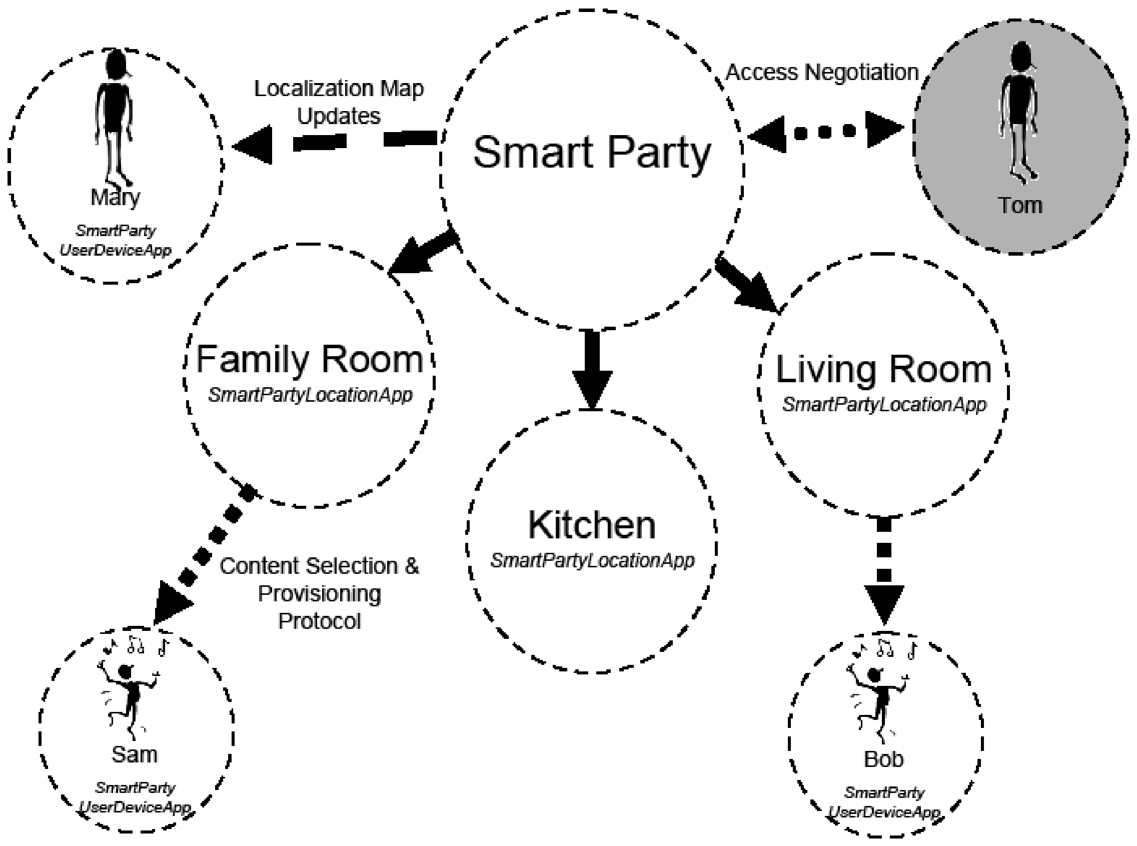


Figure 30. Layout of an in-progress Smart Party.
Shaded spheres are not yet part of the Smart Party

7.5.1 Smart Party Configuration

Figure 30 depicts the layout of an operating Smart Party installation. A communication sphere (labeled "Smart Party") acts as the master Smart Party sphere. This sphere might be one's default home communication sphere, or alternately, for security purposes, it might be a transient sphere instantiated only for the purposes of the Smart Party application. Regardless, this sphere maintains sphere relationships with the various location spheres that comprise the Smart Party. As depicted, this communication sphere serves as the central point of contact for incoming guest devices, as described above, and also provides them with localization map updates. Once guest devices are localizing within the environment, they detach from the communication sphere and re-attach to the appropriate location sphere, as determined through localization. These location spheres may run on any device in the home; however, they must be attached to speakers or have the ability to control speakers. In our deployment, the location spheres are co-located with the communication sphere, but they each control a different remote audio server, hosted on an ASUS WL500G Deluxe running OpenWRT and equipped with a USB sound card and speakers.

7.5.2 Smart Party Implementation

The Smart Party application is implemented as a Panoply application consisting of three principle components: a master application, a location-specific application, and a user device application. Together, they are implemented in 1700 lines of code, including Java boiler plate and blank lines.

SmartPartyMasterApplication: This application is launched within the Smart Party master sphere—the communication sphere for the party. The application examines existing sphere relationships and monitors the relation set for changes. When the application detects the presence of location spheres that the user has specified as relevant locations for the Smart Party, an *ApplicationLaunchEvent* is delivered to the detected sphere, where the *SmartPartyLocationApplication* is launched. Additionally, this master application is responsible for registering the localization map for the Smart Party; this enables guest devices to access the map.

SmartPartyLocationApplication: This application is launched in each location sphere. It monitors membership events, and when members are present, initiates and participates in the content provisioning protocol, as described earlier. Additionally, the location application manages the current audio play list and controls audio playback. When a room's occupant count is reduced to zero, the music in that room is faded out.

SmartPartyUserDeviceApplication: This application runs on the user device, and for this application is assumed to be pre-deployed on the guest's device along with Panoply. If necessary, the application bytecode could be attached to the invitation voucher and dynamically loaded; however, that mode of operation is not typically recommended due to the potential for malicious applications. The SPUDA manages the user's media preferences and media collection. It participates in the content provisioning protocol, expressing user preferences in the context of the current location sphere and providing media to the party at large.

7.6 Content Selection and Coordination in the Smart Party

In our previous discussion of the content provisioning protocol, we did not address the content selection algorithm, nor did we discuss the possibility of coordination among guest devices. In our initial Smart Party implementation, the SPLA processes media suggestions in a round-robin fashion—the result being that the room occupants take turns selecting content to be played. For parties consisting of only a few guests, this is reasonable and results in a decent user experience. However, when there are many guests with a wide assortment of musical preferences, we need to explore other preference management techniques, including centralized management, basic coordination mechanisms such as voting, and more complex interest-based coordination schemes.

7.6.1 Round-Robin Content Selection

In the round-robin content selection scheme, guests in each room of the Smart Party take turns providing media content. Each guest possesses a media preference vector that ranks a set of songs on a 1 to 5 scale. During the guest's slot, the SmartPartyUserDeviceApp selects the highest-ranked song that the guest has available locally. When the users in a room have heard a song, their user application may, depending on application preference, temporarily reduce the individual user's preference for that song.

When there are k individuals, round-robin turn-taking results in users hearing "their" song every 1 in k times. For fixed-length parties of s songs, each user will hear a song they suggested $\frac{s}{k}$ times. When $k << s$, users hear their songs fairly frequently; however, as the number of guests k grows large, each individual guest suggests (and thus experiences) fewer and fewer of their songs. This results in lower overall user satisfaction with the Smart Party, and as $k >> s$, a fairness problem emerges as not all users get to select songs. Additionally, there is also an *instantaneous fairness* problem with round-robin selection, as at any given moment, only one user is selecting the audio content, and he or she may be the only user who is satisfied, depending on the character of the group's preference vectors. Clearly, we would like to do better than this, if possible.

7.6.2 Centralized, Infrastructure-Based Preference Management

One obvious option we must address is centralized preference management. In a centralized preference management model, the Smart Party would centrally manage all user preferences, and automatically determine an optimal play list for each location sphere based upon the occupants' preferences. This management paradigm is problematic due to a number of issues, both technical and ideological.

Party Dynamics and Scalability

Centralized preference management would require guest devices to provide their full set of preferences to a central infrastructure node—in the case of the Smart Party, this might be per room, or per party. This has the immediate downside in that once the preferences have been submitted to a central manager, they become much more difficult to alter. Consider Alice, attending John's Smart Party. Alice has been hanging out in John's living room, enjoying listening to folk music with Bob, as per their preferences. If Alice decides that she has had enough *folk* music for the evening, and wishes to reprioritize her preferences in order to hear *rock and roll* music, she must publish a new set of preferences to the infrastructure node. Additionally, as mentioned earlier, when a user hears a song, they may reduce their preference for that song. This is a personal preference that the centralized manager will now need to maintain, or that users will need to update after every song. This process may be quite cumbersome, depending on preference representation and overall frequency of user reprioritization.

Centralized management also has inherent scalability problems. Consider that typical users of current MP3 players often carry with them music preference metadata describing thousands to tens of thousands of songs. As users will only acquire more and more content and experience with that content, future users might possess an order of magnitude or two more metadata. Additionally, content metadata is only getting richer and larger as storage capacities increase; current metadata for audio content often includes album covers and song lyrics. Together, this implies that centralized preference management will likely be increasingly infeasible.

Flexibility and Role Issues

An infrastructure-based preference management scheme is also problematic from a flexibility standpoint. Placing management functionality in an infrastructure node is a fairly rigid choice; it limits the preference management algorithm to those available at that node. As new coordination and management algorithms are made available, they will only be available within the Smart Party if the infrastructure node has been updated and the maintainer has added the new algorithms. If instead, guest devices were able to coordinate themselves, the guest devices might select any available management algorithm present among the guest device population. This operating mode is clearly more extensible.

Additionally, there is a fundamental issue regarding appropriate roles for different devices operating in a ubiquitous computing environment. Preference management directly affects the end users and imposes a burden upon the managing nodes—why should this burden be placed upon nodes which do not benefit? More appropriately, the guest user devices, whose owners directly benefit from the multimedia experience, are a more appropriate place to locate such functionality.

Privacy

Finally, centralized preference management requires user devices to provide their full set of preferences to an infrastructure component. This has substantial privacy implications for users. While voting and other mechanisms designed to elicit a communal preference typically require users to reveal some information, this is constrained and controllable by the user. Users might only reveal their penchant for country and western music after others have done so; at certain Smart Parties, proactively revealing this knowledge might result in a negative experience. Requiring wholesale divulgence of preferences may thus be considered excessive and undesirable. While divulging musical preferences may be considered a negligible loss of privacy, it is impossible to know how divulged information may be combined and then later used. Additionally, the Smart Party is a sample application that is representative of a larger application class. Other similar applications might deal with information with more obvious privacy considerations. We believe that it is better to select and design more general mechanisms that may be reused with the goal of maximizing user privacy.

7.6.3 Voting in the Smart Party

Desiring a more scalable and less-invasive mechanism than centralized preference management, we designed and implemented a voting protocol within the Smart Party in order to provide equitable content selection. Voting typically requires three phases: balloting, vote casting, and vote counting. In the balloting stage, candidates are determined and disseminated. In the second stage, votes are cast, and finally, in the third stage, votes are counted. There are numerous voting techniques that have been designed for many different systems. A full discussion of these is beyond the scope of this chapter, but [Levin1995] and [Brams2002] provide a solid introduction to the general area.

The voting scheme we have chosen is a multi-round allocation-based voting scheme. Each guest nominates a song, in the same manner as the round-robin protocol, from the available songs on the guest device. Next, the list of nominated songs is distributed to the guests in the room. The guests have a fixed pool of points that they may allocate however they choose among the nominees. This pool of points is reset for every round of voting. The voting scheme in use is multi-round—at the end of a voting cycle, points are tallied and the candidates are ordered by their point tallies. The bottom half of the ordered list is dropped, and the remaining half is resubmitted to the guests as the ballot candidates for the next round of voting. When only one song remains, it is declared the winner.

Our initial voting implementation was a classic *Borda count*, which is a preferential voting scheme where each voter must rank the candidates. After votes are collected in a Borda count-based election, points are assigned according to a predetermined mapping from a fixed pool of points. The number of points given to a candidate is determined by the number of candidates in the election. In the simplest Borda count, if there are three candidates, a candidate will receive three points for every ballot which ranks them first, two points for every ballot that ranks them second, and one point for every ballot that ranks them third. The winner of a Borda count is the candidate who receives the most points. This has the effect of electing single winners who are broadly supported.

The scheme we currently use differs in that voters may vote for a subset of the nominees, and voters may allocate points freely. The benefit of this stems from the fact that when selecting audio content, many songs are of equivalent value to the user; the user may not really prefer one to the other. Any strict ranking of these songs creates a false preference, and reduces the benefit of cooperation. For example consider User A, B, C, and D who are voting on a set of four songs.

Song/User	A	B	C	D	Raw Preference	Borda Count
Let It Be	**3** *(#1)*	**1** *(#4)*	**3** *(#1)*	**3** *(#2)*	10	12
Thriller	**3** *(#2)*	**3** *(#1)*	**1** *(#4)*	**3** *(#1)*	10	12
Sing, Sing, Sing	**3** *(#4)*	**3** *(#3)*	**3** *(#3)*	**3** *(#3)*	12	6
Sloop John B.	**3** *(#3)*	**3** *(#2)*	**3** *(#2)*	***1*** *(#4)*	10	9

Table 12. Preference matrix showing user preference for a set of songs. One arbitrarily-chosen ranking is indicated in parentheses, as well as the associated Borda Count

Table 12 describes our hypothetical users' preferences for the listed songs, and one possible ranking of these songs is indicated in parentheses. Greater preference is represented by larger preference values and lower rank. Based on the raw preferences, we see that the song "Sing, Sing, Sing" should be preferred as all users give it their top preference value of 3. If software was forced to rank the songs, it could induce false preferences, such as the ranking indicated in our table, which arbitrarily breaks ties in the user preference vector. Using a traditional Borda count, all of the other songs would have a higher score than "Sing, Sing, Sing"—clearly an undesirable result.

The use of a point system helps to mitigate this problem; additionally, our multi-round voting protocol also helps improve the overall fairness and outcome of the vote by allowing users to shift their point allocation around after songs are eliminated from consideration.

The Smart Party voting protocol builds on the Media Suggestion phase—a vote coordinator, e.g., the location sphere, coordinates the Media Suggestion phase, collecting suggestions, issuing ballots, and evaluating the results. The vote coordinator intercepts the media suggestion requests, and begins the nomination procedure with all device spheres in the location, which leverages the existing media suggestion protocol. Once nominations are complete, ballots are distributed to the device spheres. The SPUDA implements the basic voting protocol described earlier, and the vote coordinator collects the votes and handles the run-off rounds.

7.6.4 Interest-Based Collaboration

Voting is a large part of the solution, but it can only improve the user experience so much. Ultimately, the ballot is limited to the songs that the guests nominate. When guests possess large media libraries (often with tens of thousands of files), there clearly is a problem. Which songs should be nominated? How do we identify which songs are popular in a given population without a centralized collaborator? Typically users will nominate their “favorite” songs that they wish to hear at the moment. There may be many songs that meet this criterion—and many others that are also acceptable to the community at large.

The Panoply solution is distributed collaboration through the organization and formation of interest-based spheres. These spheres represent guests with overlapping preferences, or goals, who can work together to nominate and vote for their common good. Effectively, these interest-based spheres are acting as voting coalitions. Distributed coordination in the Smart Party has three main phases: coalition formation, coordinated voting, and coalition goal improvement.

Coalition Formation

The first step in coalition building is to identify a relevant and interesting existing coalition to join, or alternately, to start a new coalition. Within the location sphere, device spheres listen for coalition announcements, which contain a list of the goals of the coalition. For purposes of the Smart Party, goals are artists that the coalition is going to try to hear. Each device sphere considers the advertised goals of extant coalitions—if the goals of any advertised coalition align with the device sphere's goals, the device sphere is said to be *attracted* to the coalition. The degree of attraction is based upon how closely the coalition's goals align with the device sphere's goals. If the device sphere is attracted to any coalition—above some threshold—the device sphere will join the coalition to which it is most attracted. Our current goal model is based upon preferred artists, but goals could just as easily be represented as preferred musical genres, preferred musical tempos, etc. Individual songs could also be used, but are in general a bad choice, since goal sets would by necessity need to change fairly quickly; artists offer users more common ground upon which to cooperate. Coalition formation might also be driven by so-called experts. Individuals might team up with a coalition led by an individual with popular musical taste. This individual could effectively act as a "DJ" for the group, nominating songs that the coalition would support.

If no coalition exists, or no existing coalition is sufficiently attractive, the device sphere will start a new coalition by creating a new interest-based sphere hosted on the local device. As part of this process, the coalition founder will create a new goal set for the coalition from its own preferences. Coalition creation could be limited or otherwise restricted by the location sphere, in a policy-dependent manner. In this way, parent spheres could limit the number of coalitions for scaling purposes.

Coordinated Voting

After coalitions have been formed, coalition members act together, working toward the common goals of the coalition. During the suggestion phase of the Smart Party, members of a coalition suggest songs that are aligned with the goals of the coalition. This effectively focuses the set of available songs from which members select. In the voting phase, members allocate their points as they wish among all nominated songs, and submit their votes through the coalition voting coordinator.

Improving the Coalition's Goals

Initially, a coalition consists of one member—the coalition founder. As more members join, they will be attracted to the coalition to some degree, but in any real Smart Party, members will not be homogeneous. The coalition goals as stated by the founder will almost certainly not align completely with the goals of all members. Additionally, as songs are played, various coalition goals will be met. As a sufficient number of songs are played, new songs that meet the given goal criterion (e.g., artist or genre) may be unavailable. At that point, it is in the coalition's best interest to replace those goals with new goals, selected by the members.

To that end, we employ a goal improvement process. Coalition members nominate new goals, e.g., new artists, to replace old goals. Goal nominations are advertised within the coalition. If a majority of group members prefer the new goal to the old goal, the specified old goal is replaced with the new goal.

Intrinsic Benefits of Interest-Based Spheres in the Smart Party

Interest-based spheres provide a number of direct benefits to the Smart Party application. In terms of scalability, they reduce the number of directly connected members with which each location sphere must interact, and also can serve as a vote aggregator and concentrator. Since coordination is distributed, the overhead of coordination is divided up among the various coalition leaders. Additionally, since coalitions involve only a minimal amount of state (the coalition goal set), coalitions can easily be broken up and reformed, allowing the leadership role to be easily migrated from one device sphere to another.

Additionally, interest-based spheres assist greatly with privacy, as preferences are only partially revealed, and much of that is only within the context of the coalition. Additionally, the protocol is designed to allow each guest device to reveal as much or as little of their preferences as they desire. In terms of extensibility, functionality is pushed toward the edges. Coordination is organized by the guest devices themselves—this allows the guest devices to bring in whatever coordination mechanism they desire, and plug it into the Smart Party. Finally, since coordination does impose some overhead, it is most appropriate to position that functionality, and the corresponding overhead, at the devices which most benefit from that coordination.

Given this understanding of the Smart Party's operation, and the different possible coordination modes we have considered for the Smart Party, we turn to our measurement and evaluation of the Smart Party in operation.

7.7 Evaluating the Smart Party

This section presents our measurements and evaluation of the Smart Party. These are broken into three primary categories: operational measurements that provide basic timing data, user satisfaction with selected songs and the equitable distribution of that satisfaction, and the scalability of the Smart Party.

To evaluate the Smart Party's base operating characteristics, we ran through 1320 iterations of the Smart Party, measuring the time consumed by the various phases of the Smart Party. The client hardware used for this evaluation was an IBM Thinkpad T42 with 512 Mb of RAM, and a 1.7GHz processor. The Smart Party was hosted on a 2.5 GHz Dell Desktop PC, also with 512 Mb of RAM. Both machines were running Ubuntu Linux, Feisty Fawn release. Table 13 summarizes the data gathered from these experiments.

These measurements provide a view of the time overhead for all phases of the Smart Party, from the moment a user device detects the Smart Party environment through participation in the Smart Party media provisioning protocol. This data indicates that the user device typically detects the Smart Party environment by matching its voucher and configures the wireless network interface in less than two seconds upon

Smart Party Phase	**Mean Time 99% CI**	**Median Time**	**25% Line**	**75% Line**
Match Voucher	1373 ± 086	1233	823	2028
Configure Network Settings	334 ± 003	325	320	333
Acquire DHCP Address	8000 ± 146	7821	6317	9765
Launch Party Application and Connect to Party Sphere	2797 ± 038	2795	2449	3032
Negotiate for Access	1149 ± 018	1117	1095	1151
Receive Loc. Map from Party Sphere	852 ± 045	844	803	896
Apply Loc. Map	1959 ± 012	1955	1851	2048
Localize	16441 ± 817	12404	9896	18970
Begin Participating in Location Sphere	1776 ± 021	1734	1679	1801
Total Elapsed Time to Participation	34788 ± 827	30687	28162	37211
Total Elapsed Time w/o Localization	18333 ± 178	18039	16615	19975

Table 13. Measurements of the in-office Smart Party deployment (Data represents 1320 trials. All times are in milliseconds; means are reported with a confidence of 99%)

arriving within observation range of the Smart Party environment. Acquiring an IP address via DHCP (the DHCP client is ISC DHCP v3.0.4, triggered via JNI) consumes around 8 seconds on average, when we make the reasonable assumption that the client device does not have a pre-existing lease.

After acquiring an address, the user device launches the SPUDA and initiates a connection to the Party Sphere, and negotiates with the Smart Party social sphere for access, providing its signed cryptographic invitation voucher as proof of authorization. The combined overhead of these steps is just under four seconds. Once a member of the social sphere, the client sphere receives a localization map and applies it to its local database. This step is dependent upon the size of the localization map in question, but as shown above, the overhead is quite reasonable.

After an additional ~17 seconds, the bulk of which from localization, the device has joined the appropriate location-specific sphere and is participating in the Smart Party.

From start to finish, only 34 seconds have been consumed; these operations are completely transparent to the user. When we consider that simply entering the environment and greeting the host may, by itself, take 30 seconds to several minutes, the configuration overhead is quite reasonable. Additionally, we anticipate further refinements of the localization technology, research that is orthogonal to this dissertation, which will result in significantly reduced overhead.

Satisfaction and Fairness in the Smart Party

To evaluate the effectiveness of our various coordination mechanisms in the Smart Party at a reasonable scale, we implemented a Smart Party simulation that allowed us to create parties of arbitrary sizes and evaluate the party mechanism with real user profiles. User profiles were extracted from Last.FM, a social music website to which users can automatically *scrobble,* or submit their audio playback history, via a number of different audio tools that support the Audioscrobbler interface [Audioscrobbler]. These user profiles list the audio tracks to which users have listened, and the number of times each track was heard. We extracted 1500 such profiles from Last.FM, and mapped each user's track information into a ranking. This was accomplished by dividing the score range (*# of times most-played song was heard - # of times least-played song was heard*) equally into five buckets, and ranking songs based upon which bucket they fell into. The user data exhibited exponential characteristics, and the rank-mapping scheme we used maintained these characteristics. A very few songs had rank five; more songs had rank four, and so on. Most user songs were in the rank one and rank two buckets, as expected.

Our Smart Party simulation models a user population in a single room of the Smart Party, using varying population sizes and with different song suggestion algorithms. For each algorithm, for each population of size *k*, we run through 150 iterations of the Smart Party, selecting a new random subset of our user profiles for each iteration. The random seed used to initialize the pseudo-random number generator that generates the party populations was kept consistent between the different song suggestion algorithms, ensuring that we test the different algorithms with the same population instances and letting us effectively get a head-to-head comparison of their behavior. Finally, each simulated party is defined to be 30 songs long; assuming an average song length of around three minutes, this simulates ninety minutes of music.

To evaluate our different suggestion techniques, we require a metric. We have selected two metrics to use for measuring our suggestion algorithms: *user satisfaction,* and *fairness*.

We define a per-song and per-user satisfaction metric $U_{s,k}$ as the following:

$$U_{s,k} = 0,$$

if the song *s* is not in user *k*'s preference set, or if *s* is in *k*'s preference set,

$$U_{s,k} = 2^{r},$$

where *r* is the preference rank given by user *k* to song *s*. We use this non-linear scoring system to evaluate satisfaction to emphasize the fact that highly ranked songs are much more desirable to users than low-ranked songs, as indicated by the Last.FM user profiles. While our chosen scoring does not perfectly capture the preference dynamics in the

Last.FM user data, it does reflect the approximate weighting observed in the overall data set.

For a given round *r* of the Smart Party played, we can calculate the normalized satisfaction value RS_r,

$$RS_r = \frac{\sum_{k=1}^{n} U_{s,k}}{n}$$

where *s* is the song played in round *r*, and *n* is the size of the population. Total user satisfaction for the party, P_{SAT}, is calculated as the normalized summation of user satisfaction for all rounds in a given instance of the Smart Party. This is defined as:

$$P_{SAT} = \frac{\sum_{r=1}^{m} RS_r}{m}$$

where *m* is the total number of rounds.

For our purposes, fairness is a measure of the uniformity of the distribution of satisfaction within the Smart Party. To evaluate the fairness of the different algorithms, we make use of the Gini coefficient. The Gini coefficient, G, is a summary measure of statistical dispersion, or distribution, of wealth in a population [Gini1912]. We use the Gini coefficient to quantify the distribution of satisfaction within a given instance of the Smart Party. G measures inequality within a population; its minimum value is 0, indicating that all members of the population possess equal amounts of wealth. The maximum value for G is 1, indicating all wealth is concentrated in a single individual.

G is classically defined as:

$$G = \frac{\sum_{i=1}^{n}\sum_{j=1}^{n}\left|x_i - x_j\right|}{2n^2\overline{x}}$$

where x is the observed wealth of a member of the population, n is the size of the population, and $\overline{x}$ is the mean wealth of the population. For purposes of the Smart Party, user satisfaction from hearing a preferred song is treated as "wealth" for purposes of calculating the associated Gini coefficient.

Given these metrics, we can now turn to the simulation results. The figures below compare three different suggestion algorithms—round-robin suggestion, standard voting and coordinated-based voting, as the party size varies from 4 guests to 80 guests. Figure 31 shows the median P_{SAT} results from 150 Smart Parties simulated at each size. Error bars indicate the 25th and 75th percentile results.

These results are interesting. For small Smart Parties, e.g., parties of eight and fewer guests, round-robin song suggestion results in better overall user satisfaction, as users are able to, in-turn, select and hear their highest ranked songs. At parties of four guests both non-cooperative voting and sphere-based voting perform more poorly than round-robin in terms of median satisfaction at these levels, as both result in the selection of lower-ranked songs shared in common. However, at eight guests, sphere-based voting provides better satisfaction more than 25% of the time, and in fact, is nearly comparable to the best-performing algorithm, round-robin. As more and more users participate in the Smart Party, voting and sphere-based voting result in higher satisfaction. At size 16, sphere-based voting begins to clearly dominate—up to nearly a 50% improvement over non-

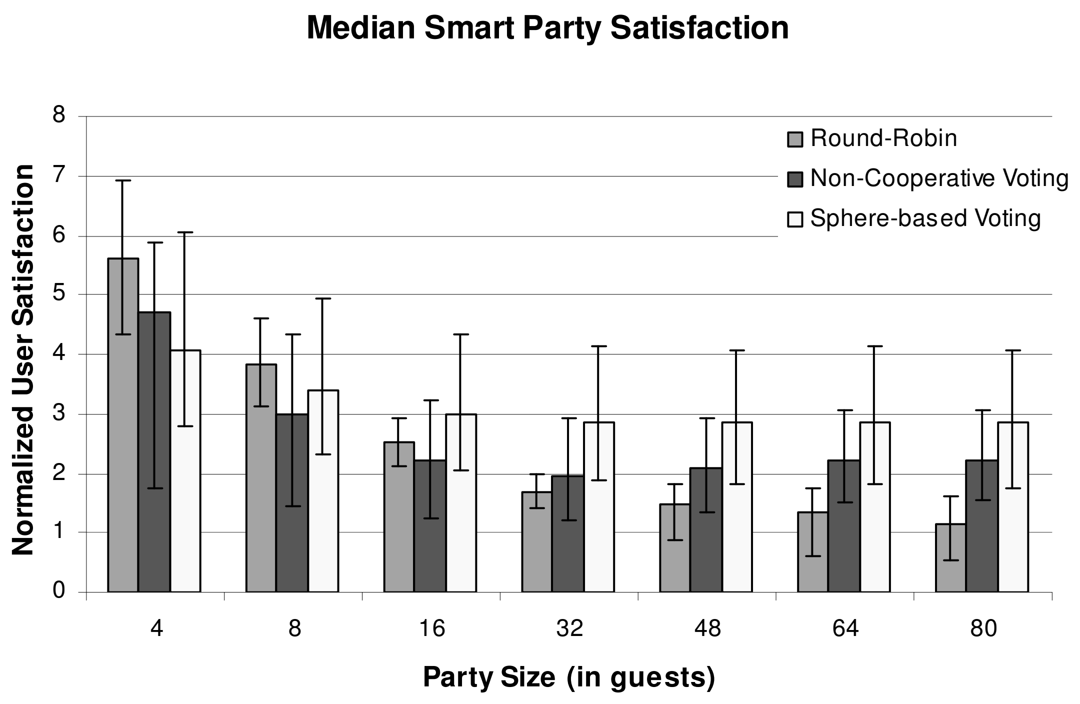


Figure 31. Median Smart Party satisfaction scores for varying size Smart Parties.
25th and 75th percentiles are also indicated.

cooperative voting at party sizes of 32 and 48. Sphere-based voting also shows a 150% improvement over round-robin at a party of 80 guests.

As more and more users participate in the Smart Party, we see an upswing in user satisfaction in the non-cooperative voting cases; however sphere-based voting still dominates. For extremely large parties, we predict that sphere-based voting and non-cooperative voting may converge in terms of satisfaction, as a very large number of songs would be suggested in the nomination phase, reducing the benefit of the coordination activities.

Next, we must consider fairness with respect to the distribution of satisfaction. Figure 32 and Figure 33 show the measured Gini coefficients for our candidate algorithms over varying party sizes. Figure 32 indicates the measured median Gini coefficient of satisfaction, when considered on a per-round basis. Figure 33 shows the median Gini coefficient of satisfaction taken over each Smart Party instance as a whole. As before, error bars indicate the 25th and 75th percentile results.

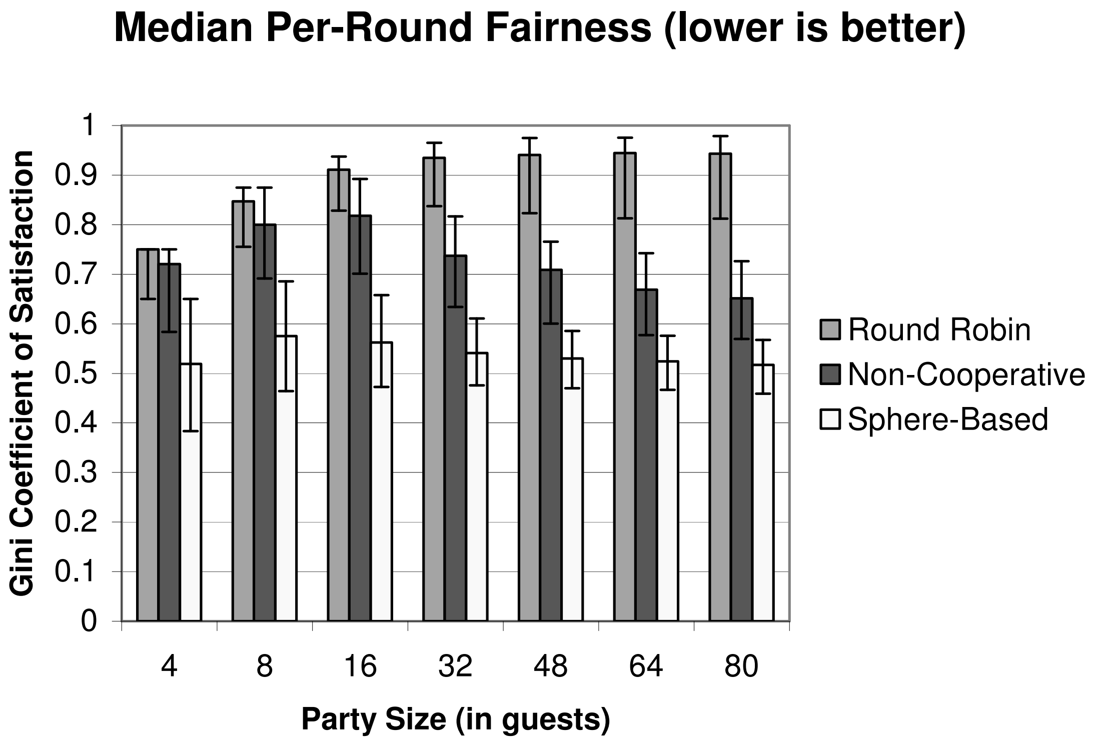


Figure 32. Median per-round fairness in the Smart Party. Fairness is expressed in terms of the Gini coefficient with respect to user satisfaction. 25th and 75th percentiles are also indicated.

These figures clearly illustrate the per-round effects of the different selection algorithms upon the fair distribution of satisfaction. When we consider fairness over entire Smart Party instances, round-robin's turn-taking results in overall extremely fair parties until the number of guests exceeds the number of song slots (30).

In terms of per-round fairness, round-robin satisfaction is principally concentrated in the suggesting guest, resulting in poor per-round fairness. Sphere-based voting performs significantly better than either round-robin or non-cooperative voting in terms of per-round fairness, and also performs better, in general, than non-cooperative voting in per-party fairness.

We believe that per-round fairness, where sphere-based coordinated voting particularly shines, is an important metric, as it captures the average user experience at any given instant with respect to his or her peers. If a user has to wait many rounds before hearing a song he or she likes, they are likely to go elsewhere.

The satisfaction and fairness results need to be taken in tandem—very fair parties with very low overall satisfaction are not interesting. For very small parties, round-robin performs better in terms of satisfaction and has reasonable per-round fairness. For parties of eight or more users however, sphere-based coordination performs as well or better than the other algorithms in terms of user satisfaction, and is better across the board in terms of per-round fairness.

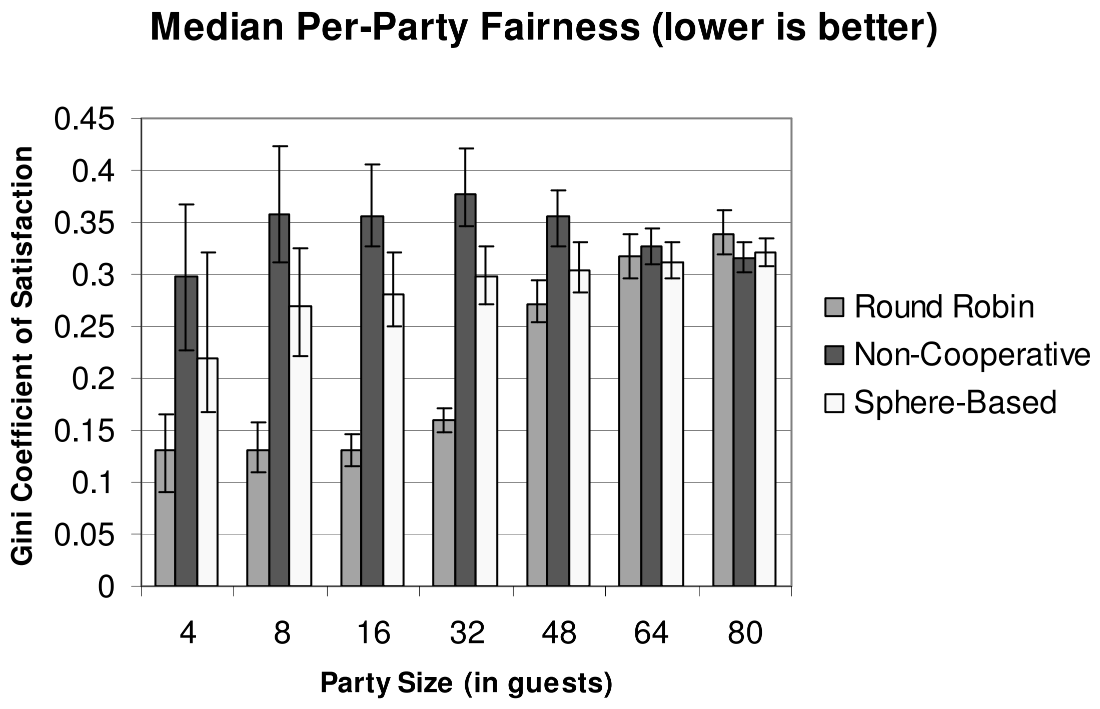


Figure 33. Median per-party fairness in the Smart Party. Fairness is expressed in terms of the Gini coefficient with respect to user satisfaction. 25th and 75th percentiles are also indicated.

In practice, a reasonably sized party at someone's home might fall between 8 and 32 guests. Other venues with much larger populations might also use Smart Party technology—such as a dance club, or an office party. In both of these scenarios, sphere-based coordination is clearly superior.

Scalability and the Smart Party

Finally, we are interested in the scalability of the Smart Party as the number of guests increases. To evaluate the scalability of the Smart Party and the spheres of influence

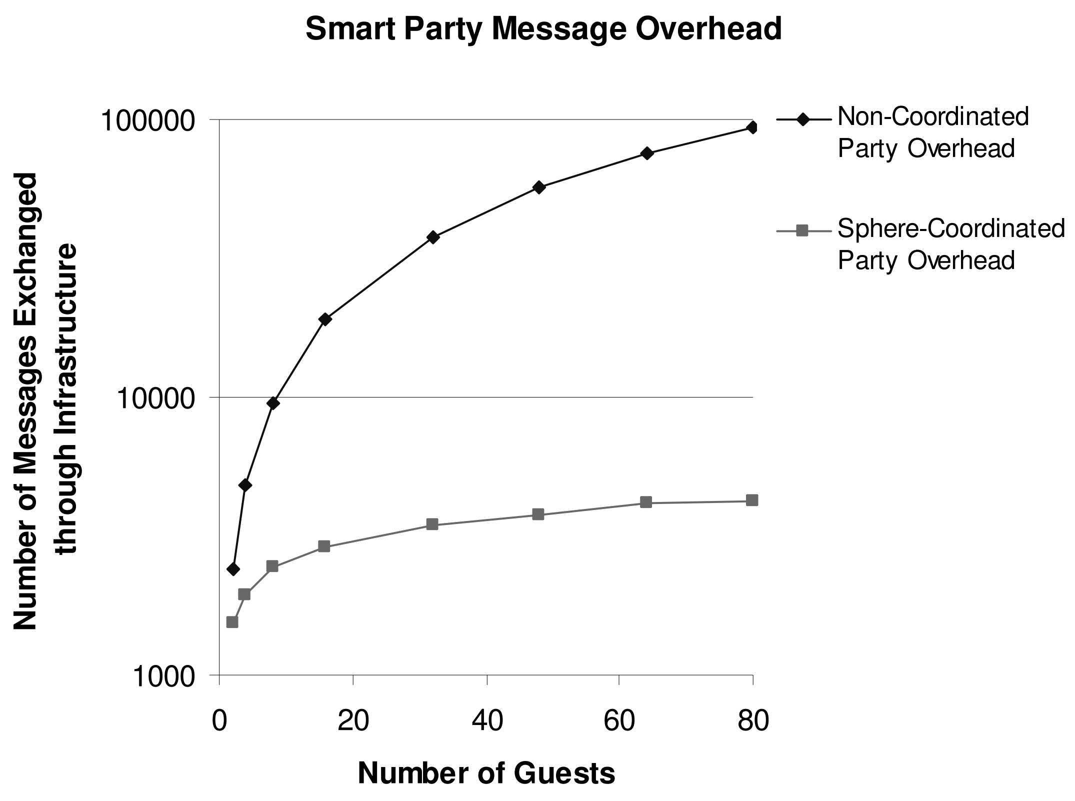


Figure 34. Scaling the Smart Party.

model of coordinated grouping, we measured the overhead incurred by the location infrastructure in terms of message count. We compared the non-coordinated mode of Smart Party participation to participation that was coordinated through interest-based spheres. Our overhead measurements included all messages passed between guests and the location infrastructure. Specifically, this includes the voting protocol and the media provisioning protocol, as well as Panoply's sphere maintenance protocol.

Figure 34 summarizes the behavior of the different operating modes in terms of incurred overhead. Note that these results are represented in log-scale. We can clearly see that sphere-coordinated participation in the Smart Party is much more scalable as the number of guests at the Smart Party increases. The scalability gain is due to two main factors. First, individual coordination groups are handling much of the ballot distribution and vote gathering internal to the coordination group, and passing aggregated responses back to the location. Additionally, the location infrastructure only has to maintain connections to the group leaders, instead of to every guest device, providing a substantial reduction in message overhead.

7.8 Wrap-Up

The Smart Party demonstrates end-to-end configuration of a group of user devices, with minimal user intervention. Within the Smart Party, guest devices are taken from an unconfigured state where they only possess a "party invitation" all the way through membership in a coordination group within the appropriate room within the party environment and network, with virtually no required human interaction.

Panoply greatly simplified the development of the Smart Party application. The basic Smart Party application was designed and implemented in Panoply in approximately two weeks by three developers; Panoply provided much of the basic functionality necessary for the application, including basic network configuration and voucher-based discovery, location map dissemination and localization, and group discovery and scoped communication primitives.

Interest-based spheres group together user devices with common preferences, amplifying the overlapping preferences of the guest devices, and allowing common preferences to emerge more easily. This results in an improved Smart Party experience in terms of total user satisfaction, fair distribution of that satisfaction, as well as scalability.

We believe that this application is both exciting and forward-looking, given current trends in portable mobile devices and social media. Enabling the formation of interest- and preference-based coalitions will be increasingly important as more and more individuals wish to interact simultaneously with environmental services and applications.

Chapter 8

Related Work

A number of projects exist, both in research and industry, which are relevant to Panoply and the set of Panoply applications and services that we have developed. This section breaks these down into the following categories: general ubiquitous computing infrastructure, device discovery and configuration systems, event-based systems, group-based systems, mobile security, and finally, social media projects.

8.1 General Ubiquitous Computing Infrastructure

Panoply is a general-purpose ubiquitous computing middleware, and bears some similarity to other general-purpose ubiquitous computing infrastructure projects in that it shares a high-level goal of enabling novel applications and system services, often within non-traditional computing environments. This is typically accomplished by gathering and providing information about the user, his environment, and other system context that can be leveraged by applications. Ubiquitous computing environments have been explored in various research contexts for the last decade. Numerous projects have examined augmenting physical locations with numerous devices that react to human action. Notable projects include the Intelligent Room [Brooks1997], Centaurus [Kagal2001a], Gaia [Román2002], and one.world [Grimm2004a,Grimm2004b].

The MIT Intelligent Room and Project Oxygen

The MIT Intelligent Room project [Brooks1997] focuses on enabling computation to adapt to the environment and users through the use of a minimal set of infrastructure-based sensors and actuators, including steerable cameras, microphones, projectors, video-multiplexers, and electronic lighting controls. Using this set of input and output devices, the Intelligent Room tracks users as they move around and attempts to identify ongoing tasks, assisting where possible. The Intelligent Room system focused on task identification, and recognizing user gestures such as shaking, pointing, etc.

The original Intelligent Room has been extended with a number of other projects under the umbrella of Project Oxygen. The Intelligent Room now allows the user to specify resource and service requests in high-level terms [Gajos2001, Hanssens2002]. The infrastructure is responsible for interpretation of requests and allocation of resources by means of an intentional naming system [Adjie-Winoto1999]. These decisions are made at run time using information about current user location and tasks. In the original model, security and policy management is minimal, with only basic access control provided for resource access. Component interaction in the Intelligent Room is organized through Metaglue [Coen1999], a Java language extension that adds language support for interconnecting, controlling, and adding components in the Intelligent Room via a centralized component registry.

The Metaglue component model was extended by Hyperglue [Peters2003], which is designed to enable interactions between services and devices in multiple active spaces and remove the need for centralized management. Interactions are managed by a DNSstyle service lookup. Hyperglue allows Intelligent Rooms to manage themselves independently and interact with others as a single virtual entity. It also augments the Intelligent Room with a role-based access control scheme [Kottahachchi2004]. Access to the system is limited to entities that can prove a transitive relationship to an authorized entity.

The Panoply model differs in several important respects. Most importantly, the Intelligent Room focuses on resource management, while Panoply's focus is on configuration and connection management. Within Panoply, devices are considered as autonomous entities, not extensions of their current location. This allows our more general notion of the computational group context in which a device participates—the Sphere of Influence. Additionally, within Panoply, authorization is based on both actions and identity. This allows for our so-called "interactions with strangers." In other words, foreign devices may interact with Panoply devices, assuming the actions they wish to undertake are allowed by policy.

Project Centaurus at the University of Maryland

Centaurus [Kagal2001a] provides an infrastructure and communication protocol for interoperation of heterogeneous mobile devices. The Centaurus architecture attempts to address the needs of a typical *Smart Room.* A Centaurus enabled-environment consists of a set of communication managers, service managers, clients and services, each unit or Smart Room having a communication manager and a service manager. Vigil [Kagal2002b], an extension of Centaurus, layered some basic security features on top of the Centaurus model. Within an environment, certificate controllers generate and assign digital certificates to requesting entities, while a security agent maintains trust information for validation and revocation. Interaction between Smart Rooms is not described, other than through a hierarchical arrangement of service managers within a single domain. Vigil associates a static set of rights with the role a device can assume, which does not allow devices to acquire privileges dynamically. Vigil does not deal with device or environment integrity management.

The follow-on to Centaurus, Centaurus 2 [Undercoffer2003] adds further security and interoperability mechanisms. Users in one space can access a service in another using this hierarchy. Centaurus 2 elevates security to a principle concern. Authentication and access control is handled through a centralized PKI.

The Smart Spaces model used in Centaurus and Centaurus 2 is similar in kind to the Panoply Spheres of Influence model when we restrict our attention to location-based spheres. Of course, Panoply also supports the creation of contexts based on communication abilities, social groups, and other common application interests. Additionally, neither the Centaurus security model nor the Centaurus 2 model support dynamic membership, which is necessary for most applications. Access is limited to preauthorized clients who are granted static privileges to interact with a predetermined set of services.

Gaia at Carnegie Mellon University

Gaia is a distributed middleware infrastructure designed to coordinate heterogeneous software and device entities in ubiquitous computing habitats or living spaces. Gaia's main focus is on context-aware resource management in homes, offices, and meeting rooms. Gaia provides application-aware adaptation facilities to dynamically adjust applications to changes in the Gaia ActiveSpace, access remote data, and dynamically suspend and restart applications in new contexts. Similar to the Intelligent Room, Gaia assumes the existence of a large variety of sensors and actuators supporting the environment that will sense user behavior and assist with tasks. Gaia views itself as an operating system that provides management services for coordinating distributed objects used by applications.

Mobile Gaia [Chetan2004] is a personal cluster version of Gaia, intended to provide resource management and sharing within an ad hoc personal device cluster. Mobile Gaia provides basic cluster management for the personal cluster. When a cluster "coordinator" detects a new device, it extends an invitation to the new device to join, which the device may accept.

Another Gaia sub-project, Super Spaces [Al-Muhtadi2004] is a proposed management framework for managing large-scale environments composed of multiple Gaia Active Spaces. It proposes to extend the ActiveSpace paradigm through Super Spaces which connect and allow ActiveSpaces to coordinate and share services. Few technical details are available regarding the implementation and actual functionality of Super Spaces.

Superficially, Panoply and Gaia share several features; however, all of the various Gaia incarnations focus on managing devices and resources within a single administrative domain—either location-based, or a personal cluster of devices under a single user's control. The Panoply model allows the representation of diverse communities of devices—whether in physical environments, personal clusters, or social contexts.

Additionally, Gaia is focused squarely at being a “ubiquitous operating system” in terms of resource management. Panoply, in turn, takes a minimalist approach by providing a few key services in order to enable application-directed management of personal devices.

one.world at University of Washington

Similar in some ways to Gaia, one.world [Grimm2004a, Grimm2004b] also attempts to be an operating system for ubiquitous computing, managing resources, devices and applications within a single domain. one.world takes the approach that change is inherent in ubiquitous computing environments, and change needs to be exposed to applications to encourage ad hoc composition and enable sharing. one.world focuses on supporting application relocation and migration using virtual machine-based environments, and managing resources for applications.

We agree with one.world’s high-level goals; however, Panoply differs from one.world in several important ways. Again, Panoply focuses on providing a small set of necessary services to simplify application development and device management, but without requiring an entirely new operating system. one.world assumes devices possess suitable low-level network configurations to interact; Panoply makes no such assumption and handles low-level network configuration issues for mobile devices. Additionally, Panoply’s event-based and device-oriented approach means that any device speaking the appropriate protocols can participate in Panoply, regardless of runtime environment or internal application model, while one.world restricts applications to a specific set of functionality and an operating environment that handles relocation and migration.

Additionally, one.world does not provide a security model for its operational paradigm, nor does one.world provide any security primitives for applications to build upon.

8.2 Discovery, Configuration, and Connection

Industry has focused its efforts on simplifying the process of connecting devices to one another, principally in the arena of the Small Office/Home Office (SOHO). We will discuss a variety of efforts that have emerged to address automated configuration of local-area devices, so-called "zero configuration," including Microsoft's Universal Plug and Play and Apple's Bonjour, formerly Rendezvous [Apple]. Also related to these efforts, as well as to Panoply, we will discuss Sun Microsystems's Jini network architecture.

Universal Plug and Play [UPnP] is designed to address automatic configuration and connection of network devices. UPnP introduced the Simple Service Discovery Protocol (SSDP) which utilizes locally scoped multicast groups to identify local services and resources. Devices can advertise their capabilities using SSDP and connect to one another to query device status and configuration through SOAP (Simple Object Access Protocol). In practice, UPnP is used to identify and connect personal computers to printers and scanners, set up port forwarding in personal firewalls, etc.

A similar technology, Apple's Bonjour, is a general-service discovery protocol suite that utilizes link-local IPv4 (IPv4LL) [RFC3927] to automatically assign IP addresses and multi-cast DNS [mDNS], among other protocols, to locate printers, file servers and other services on a local-area network. Panoply takes a similar approach to sphere discovery within a LAN, using locally scoped multicast announcements, as well as broadcast UDP beacons, to discover and identify relevant spheres. Additionally, Panoply uses IPv4LL techniques to handle automatic address assignment for ad hoc sphere formation. Panoply differs characteristically from both UPnP and Bonjour in that these are wholly configuration protocol suites designed to connect local-area services and applications. Panoply supports direct application communication with peers through our event-based communication framework. Panoply also handles wireless network selection, configuration and join mediation, group discovery, and connection management.

Another interesting effort, Sun's Jini [Waldo1999], is a Java-based technology intended to enable autonomous service discovery and resource access. Jini is intended to be easier to use and maintain than similar frameworks like CORBA or DCOM, where changes to protocols must be universally implemented. Devices can register and discover services through a local-area lookup service and access them through a mobile proxy—a small piece of mobile code that implements a common interface and translates calls to the interface API into implementation-specific protocols and instructions. This supports spontaneous interoperation, since every device in a Jini-enabled space speaks Java and communicates through Java RMI. While code isolation approaches such as Java sandboxing and other forms of virtual-machine isolation can help mitigate the risk, there are clear trust and safety issues inherent in the Jini mobile proxy model, due to the risks of executing untrusted code.

Panoply resembles Jini insofar that both possess discovery and interoperability components. Panoply and Jini both use multicast discovery to locate relevant peers.

Panoply uses this to discover related or interesting spheres with which to join and form groups for the purposes of collaboration and coordination. Jini allows applications to discover specific network services such as printers, file shares, or any other describable service. The Jini mobile proxy allows spontaneous interaction; Panoply also strives to allow spontaneous interaction, but instead of uploading mobile agents to devices, Panoply uses sphere join mediation to allow unknown devices to establish service agreements and privilege, identity, and membership requirements.

8.3 Publish-Subscribe Systems

Much work has been done in the general area of event-based systems, particularly in application to the special needs of mobile systems. Describing the breadth of that work is beyond the scope of this section, and unnecessary, as we do not claim any particular novelty with regards to the general Panoply event delivery system.

The one exception to that is the scoping system which Panoply employs to allow applications and application designers to limit the propagation and thus visibility of application events. This work is related in kind to the scoping mechanisms presented in [Fiege2002]. Fiege et al. integrate scoping mechanisms with the REBECA event-based e-commerce architecture [Fiege2000]. This work mainly focuses on the engineering challenges inherent in implementing scopes in such an e-commerce system. Fiege provides examples that illustrate how scopes can be used to connect heterogeneous systems intended for stock purchase and notification. In that context, event scopes limit propagation of stock orders and acceptance notifications to the "appropriate" channels based on tagged “session” scopes. Panoply explores event scopes in a different direction: the notion of a session scope is most closely related to our directed events, which allow events to be directed to a specific sphere, or set of spheres, given a set of sphere relationships. Session scopes could be implemented in Panoply, either on top of directed events, or trivially, as a new pluggable event scope. Panoply’s contribution to event scopes is the exploration of appropriate scopes to support ubiquitous computing applications. We have developed a number of useful scopes that use Panoply’s sphere relationships and sphere type descriptors to limit event visibility and propagation. This improves application scalability and provides developers with a flexible system for customizing event flows.

8.4 Group-Based Systems

Much research has been conducted in the general area of group-based distributed systems, and most of this work is far outside the scope of this discussion. We will focus this survey on dynamic group-based systems used to support ubiquitous computing with location groups, social groups, and interest-based groups.

Supporting Dynamic Location Groups

Large ubiquitous computing infrastructure projects typically involve organizing location-constrained groups of device. The systems discussed in Section 8.1 all offer a basic notion of location-based groups, where location is determined in generally rudimentary ways, either via static configuration or through wireless network proximity. In addition to these infrastructure projects, there are other related works that require discussion.

One relevant project is AGAPE (Allocation and Group Aware Pervasive Environment) [Bottazi2003], a middleware designed to support location-based group membership management. AGAPE provides basic primitives to support what it terms “network localities,” defined by the transmission range of a wireless base-station. Within a network locality, multiple operational domains may be defined. The AGAPE model is similar to our spheres of influence in that AGAPE entities are divided into structured groups to simplify management and organization. AGAPE also offers basic authentication facilities to restrict entry into a domain. Panoply differs from this effort in multiple dimensions; most importantly, Panoply supports discovery and group formation based not only on location and on network communication, but also based on social groups and application interest. Panoply also supports dynamic device configuration, and offers a rich set of security and policy features. Additionally, Panoply’s 802.11-based localization model is much more refined and extensible than AGAPE’s “network locality” based model.

Another effort related to Panoply location support is [King2007]. King et al. developed a “fingerprint distribution system” for 802.11-based positioning. This work focuses on the dynamic selection of a set of fingerprints, a set of 802.11 access point and signal strength observations, and distributing these to mobile devices. This is similar in kind to the localization map dissemination framework that is part of Panoply. The focus of this work is to improve the scalability of fingerprint-based localization systems by only requiring that mobile devices possess the small subset of fingerprints relevant to their actual position. This work addresses localization within a single administrative domain. This differs from our work in that Panoply offers a secure distribution and location configuration service, providing localization maps to support devices entering into new locations, and potentially into new administrative domains.

Supporting Dynamic Interest-Based Groups of Devices

One application of Panoply is that of supporting the discovery and formation of device communities interested in a common task or participating in a common application. Panoply provides basic primitives to support discovery and connection management of these interest- or task-based groups. Much research in task-based computing has been carried out in the realms of grid computing and mobile ad hoc networking. Existing work focuses on decomposing tasks into distributed jobs, discovering relevant nodes and resources, node selection, job assignment, task management, and so on. Largely, this work is done in the context of large, federated systems, such as grid-based systems, or sensor network arrays, or in heterogeneous mobile environments where nodes may have very different functionalities.

The Panoply effort is not intended to compete with existing task-based computing work. The Panoply model for group discovery and formation is a simple, identifier-based scheme where devices perform discovery and advertisement based on application-specified identifiers, or interests. This is intended to minimize complexity for developers and simplify group management. Clearly, more advanced task management facilities could be layered on top of this functionality if needed. Our applications have not yet required more sophisticated task management facilities.

Within Panoply, instead of allocating resources and tasks among nodes, our interest-based groups act as computational contexts for improving the group's application experience. Whether it is assisting dynamic song selection in the Smart Party or sharing location metadata in the locative media application, Panoply allow devices to leverage the knowledge and facilities of other cooperating devices to assist one another. This type of cooperation is not new, but is currently receiving increased attention.

The most well-known example would likely be BitTorrent [BitTorrent], a peer-to-peer file-sharing system. Initially, a “seed” begins to disseminate files to peers in chunks. As peers accrue more and more chunks, they begin to disseminate chunks to one another, reducing the load on the seeds, and increasing total network utilization. BitTorrent uses a “tit-for-tat” upload scheme in which peers receive prioritized service based on the degree to which they participate. Participation is based on pair-wise transfers between peers, on a per-file basis. The BitTorrent model of cooperation directly influenced our thinking about coordination in Panoply; the synergistic benefits possible from coordinating groups are clearly illustrated by BitTorrent.

BlueTorrent [Jung2007] investigates the use of cooperation among nearby Bluetooth-enabled mobile devices to download large files using a BitTorrent-like model. The work focuses on empirically determining optimal protocol parameters to support this type of application. Parameters include Bluetooth neighbor discovery frequency and peer selection. The analysis performed in this study is invaluable; similar analysis could be performed in the context of Panoply’s discovery and rescue support mechanisms to determine optimal settings for different applications and mobility models. This approach is also related to our ad hoc rescue mode that is offered by the Locative Media application, in which a nearby device may be consulted for location metadata if infrastructure service is unavailable.

The Personal Network Federations [Hoebeke2006] work being undertaken as part of the MAGNET Beyond project [MAGNET] outlines a set of requirements for goal- or service-based coordination among members of disparate personal networks, but not a specific approach or architecture. The system requirements include secure "federation" creation and management, an inter-communication security model, application support, and context support for automatic federation creation. We agree with the authors as to the basic system requirements and directions. Moreover, to varying degrees, Panoply addresses the requirements laid out by Hoebeke et al. Panoply successful demonstrates the principles of interest-based coordination; the Smart Party illustrates how this can be done securely, and in response to environmental context, e.g. location and user applications.

Supporting Dynamic Social Groups and Social Context

Increasingly, context-based systems are being used to support social awareness; we need look no further than Google's Dodgeball [Dodgeball], a text messaging-based system that allows friends to share their location via a centralized service. Users use their mobile phones to "text" their locations to the Dodgeball service, which then disseminates the location information to interested recipients among the users' social networks. A related approach, Reno [Smith2005], built on PlaceLab [LaMarca2005], also supports social disclosure of location. Friends and family members may elect to automatically disclose their location, real or fabricated, to social peers via a cell-phone-based service. The use of a localization infrastructure provides common reference points for establishing a common understanding of place.

The next step is to take this social context as input to a larger group-aware software system.

The AWARE architecture [Bardram2004] supports context-mediated social awareness among a group of users. The AWARE architecture provides a context service through which users can publish context such as their activities, location, environmental inputs, etc. These are disseminated using a centralized message service among social networks as defined by user "contact lists." This has been used to implement an "AwarePhone," a mobile phone-based application designed to improve situational awareness in a hospital. The AwarePhone application displays the current activity and location context of individuals in the phone's contact list, enabling doctors and nurses to identify peers with whom it is convenient to collaborate.

Going further, Wang et al. [Wang2004] investigated persistent social groups within the context of ubiquitous computing environments. In this research, user groups are assumed to have regular, recurring social interactions. These are codified, together with application context and limited location context, and distributed among the relevant user devices. These group profiles are then used to trigger a group management protocol at the specified time and location to discover and build groups of user devices for purposes of engaging in a group application, such as group chat.

These approaches focus on maintaining a group awareness of the different contexts in which members of a social network are active, in terms of user activity and location.

Wang et al. take this further by explicitly using user group profiles as input to a group management system.

While related to these other projects, Panoply is a very different type of system altogether. Panoply operates at a lower level than these other technologies, offering services that they do not, or cannot, provide. These include connectivity management, a mediated network join process, and rich application communication support.

While Panoply does not explicitly offer a user-level social network model, such a feature could be added as an extension to the Panoply framework, or could be implemented as a component of a Panoply application using existing system features. Panoply offers sphere discovery and organization primitives that applications can use to automatically discover and organize into a device community corresponding to application-level social relations.

8.5 Mobile Security

The Panoply admission control model led to the development of QED, a security model for protecting mobile devices and their supporting infrastructure. QED [Eustice2003b] first applied the paradigm of network admission control to mobile devices. We believe that QED is the first documented instance of a system designed to address the entire end-to-end problem of device quarantine, system examination, and decontamination for mobile devices. Additionally, to our knowledge, we are the first to publish analytic results illustrating the performance of a network access control system aimed at slowing the spread of network infection among a population of mobile devices [Anderson2005].

Since 2003, a host of network access control (NAC) products have emerged. As of November 2007, there were more than twenty different NAC products that were either available or announced [NAC]. This includes products by Cisco, HP, Juniper Networks, Nevis Networks, Sophos, Symantec, and many others.

The most well-known system in this area is the Cisco Network Admission Control (C-NAC) [Cisco2003], which restricts access to a network based on identity or security "posture" of a device. C-NAC is infrastructure-based, and can force network-level user and device authentication, as well as determination of system compliance, before a device can connect to a network. C-NAC also supports a quarantine network area where devices can undergo a remediation process before being allowed to reattempt to join to the network. A device will be considered to be compliant if it has the latest patches as recognized by a network policy server, which then makes the appropriate admission control decision—permit, deny, quarantine, or restrict. Compliant Cisco routers enforce access control, preventing non-compliant devices from communicating with the rest of the network through access control lists. C-NAC is a dedicated hardware solution that has been added to recent Cisco switches and routers.

Recently, a sub-group of the Trusted Computing Group has formed to define an open "Trusted Network Connect" architecture [TCG-NC]. This will include a set of standards for end-point integrity, and protocol specifications to allow for cross-network interoperability.

8.6 Socially-Driven, Location-Aware Media

Two of our Panoply applications, the Locative Media Application and the Smart Party, are instances of social locative media applications in which Panoply and Spheres of Influence are used to present media to users. The Locative Media Application uses an explicitly defined team context to customize the locative media experience, while users within the Smart Party environments interact with one another via coordinated preferential voting to influence a shared media experience.

There are a number of projects relevant to our locative media and social music applications; the following sections will discuss these efforts.

8.6.1 Locative Media

Mobile Bristol [Hull2004] and InStory [Barrenho2006] are both toolkit-based approaches that support the authoring and deployment of locative media. Mobile Bristol focuses on enabling rapid authoring of locative media contents, or *mediascapes*, on Windows-based PCs and palmtops. InStory provides an authoring environment that supports mobile storytelling, gaming activities and access to relevant geo-referenced information through mobile devices such as PDAs and mobile phones. The infrastructure provides localization services, as well as relevant media encoded in XML, similar to Panoply. InStory also allows users to interact with media via a graphical interface. Mobile Bristol and InStory primarily focus on enabling easy content development by authors with limited programming skills. Another related toolkit, the iCAP toolkit [Dey2006] allows users more control over how their designed applications behave without having to write code, though it does not provide infrastructural features such as localization.

Panoply adds to the richness of the experiences that can be created by these toolkits by providing group primitives supporting discovery, communication, and advertisement. Additionally, these other systems either assume no connectivity, or take it as a given. Panoply manages dynamic network selection and configuration, a hard problem that is crucial to the success of mobile applications that need to interact with other local wireless devices that may be behind local firewalls and inaccessible via metropolitan area wireless connections.

Other types of interactive entertainment, including the Ghost Ship [Hindmarsh2002], Pirates! [Bjork2001], the Seamful Game [Boriello2005], CitiTag [Quick2004]) and a variety of digitally augmented museum tours [Chou2005, eDocent, Kwak2004, Schmalstieg2005], have also explored digitally mediated user interactions in physical spaces. In these various projects, users play games or explore objects within their environments through mobile devices or devices integrated into the environment.

Each of these are specific applications or installations; Panoply could be used in most cases to implement or extend the existing applications with better network management and support for group interactions.

8.6.2 Social Music Applications

There are several projects related to our Smart Party application that bear mentioning. The most related work is MusicFX [McCarthy1998] and FlyTrap [Crossen2002], which both explore social music experiences that adapt to user preferences. They focus on equitable music selection within a shared environment. MusicFX and FlyTrap both use RFID-based detection of user badges to determine user presence, and then activate user agents on a centralized server that represent the users and vote on their behalf for tracks (FlyTrap) or music channels (MusicFX). Follow-on work to MusicFX [Prasad1999] investigates the use of an economics-based model to improve the fairness of overall channel selection. Extra vote weight is given to individuals who are forced to listen to non-preferred music. The economic scheme proposed by Prasad et al. could certainly be adapted to the Panoply Smart Party, and would provide another means to improve overall fairness within our Smart Party. In general, however, our Smart Party application differs from all of these in several ways. We support a Smart Party consisting of multiple environments with dynamic membership. Members may bring in new music via their mobile devices, and members coordinate their voting based on dynamically built preference groups. Additionally, our Smart Party is designed as a next-generation consumer application. We have proposed and implemented end-to-end configuration and connection management solutions that are not considered in the related work. Existing work assumes that the user base is static and global user preferences are preset.

Additionally, a group at the International School of New Media at the University of Lübeck in Germany has developed what they refer to as "Campus Party," or the "Smart Party" [Nikolova2007]. The Campus Party also uses RFID-based user detection, as well as a centralized database of user preferences and static music repository. The Campus Party uses a simple preference evaluation heuristic to select the current "best" song based on current occupancy of a room. The details of their selection algorithm are not described in their paper. Again, we differ from this work in terms of our support for multiple connected environments, coordinated voting within preference groups, dynamic membership, etc.

Relevant, but less immediately related, Jukola [O'Hara2004] is an interactive MP3 Jukebox device that allows a group of people in a public space to choose the music being played via a democratic process. Members in each group share a single handheld device and confer orally to find a common vote. The Panoply Smart Party makes this process completely transparent to the users.

Chapter 9

Future Work

There are a number of different directions that this work can take that will continue to make Panoply more useful, both as a general ubiquitous computing development tool, and as an active configuration and connection management system. This section discusses these possible future directions. There are several potential areas for future work. First, we will discuss a number of primarily engineering-oriented infrastructure improvements to Panoply. Next, this chapter will explore a more theoretical refinement of Panoply and the Spheres of Influence model. Finally, we will discuss possible future directions for Panoply applications.

9.1 Infrastructure Improvements

Panoply currently provides a rich set of features. However, as we developed applications with Panoply, we noted a number of features that could be added or improved to better support pervasive computing applications. In particular, we note improvements that can be made to the Panoply event system, to general reliability, management of social spheres, further development of cooperation support, and better support for resource-constrained devices.

9.1.1 Event System

There are number of improvements that would be interesting to explore in the Panoply event system, including event delivery feedback and improved event scoping.

Event Delivery Feedback: Currently, there is no feedback mechanism within the event processing system. Event senders do not receive any notification regarding the success or failure of event delivery, nor do they receive any information regarding the identity of event receivers. Supporting some form of event delivery status notification would be valuable to event-generating applications and Panoply components. Applications would know when control messages or data were not received, and these notifications would simplify the development of reliable event transport, either in the application or as a Panoply service. Additionally, knowing the nature of the recipients would allow Panoply applications to customize event handling further. Interesting context to return to the event sender could include the type of recipients, their number, and their relationship to the sender. One example application of notifications would be the ability to send an "initial" probe event to determine if the listening event receivers were of use to the application. If the application determines that no interesting event receivers are available, it can provide informative feedback to the user, or attempt a different operation.

Event Scoping: In addition to providing event feedback, we would like to improve and extend the existing event scopes. First, we would like to develop a new scope that we refer to as *nearest interest scope*, as it directs an event to an interested recipient who is the fewest hops away in the Panoply relationship graph. In certain applications, such as location-based applications, we frequently wish to deliver an event to the single "closest" interested subscriber. This type of scoping requires new functionality that Panoply does not currently possess. Specifically, we need to track relationship distance along with event interest registration. One can imagine extending the functionality of this event scope further, by limiting the traversal of relationship links based on role, such that only parents could be included in the calculation, or only children, etc.

In addition to this nearest interest scoping, we would also like to add support for composing event scopes. This would allow different types of event scopes to be combined. Combination possibilities would include unions, e.g., scope X OR scope Y, intersections, e.g., scope X AND scope Y, as well as sequence combinations, e.g., scope X THEN scope Y. This would allow much richer event scopes, particularly if composition scopes could recursively include themselves. It is important to note that such a feature might open up a potential vulnerability in Panoply. One possible denial-of-service attack on Panoply would be to attach extremely complex composed scopes. If such a feature were added, it would be important to evaluate its performance in the face of complex event scopes, and mitigate the threat of denial-of-service through the scope composition mechanism.

Event Interest Scoping: Another improvement we envision to the Panoply event system is the addition of event scopes to event interest. Currently, event interest is not scoped. Event interest flows between peers unrestricted. In sufficiently large and sufficiently dynamic arrangements of spheres, this will inevitably cause scaling problems. Additionally, spheres may wish to limit the propagation of event interest for other reasons, such as relevancy. For instance, an application may only wish to receive events of type X that originate in location spheres, etc. To that end, it would be valuable to add event scoping to event interest messages. However, unlike individual events, the event interest expressed by a sphere may stem from multiple different applications or spheres. As a result, there may be multiple different scopes that would need to be applied across a given relationship. Each sphere handle managed by the relation manager would need to specify the set of scopes relevant to the event types that the remote sphere is interested in, and filter each incoming event appropriately. This would need to be handled efficiently, so as not to introduce substantial overhead into the event processor.

9.1.2 Fault Tolerance & Reliability

The second general area of improvement for the Panoply infrastructure would include improved fault tolerance and reliability support. There are two main thrusts for improvements in this area. The first is sphere leadership failover, and the second is support for disconnected nodes.

Sphere Leadership Failover: Currently, we support automatic failover for sphere leadership for interest-based and communication-based spheres via a deterministic election scheme. For simplicity, we will refer to these spheres as composite spheres, as they are composed of three or more other spheres. When the leader of one of these election-based spheres goes away, the formation algorithm restarts. The individual device spheres reconnect to one another and elect a new leader. Meanwhile, events sent while the interest sphere is re-forming may be lost, or may not be delivered to all interested recipients.

To reduce event loss and speed up the failover process, we can improve the leadership model through the use of tiered leadership. Once the composite sphere has been formed, a leadership hierarchy can be established, e.g., the leader can appoint a second-in-command, a third-in-command, etc. This could also be accomplished through the election protocol itself. If all members of the composite sphere know who the new leader would be in the event of leader failure, failover could be greatly sped up. Additionally, the members of the composite sphere could maintain connections to more than one member of the leadership, possibly even duplicating, or "carbon-copying" events to the sub-leader(s). In the event of leader failure, the sub-leader could immediately take over. Properly implemented, this could greatly reduce the loss of events due to leader failure. The size of the leadership could also be a dynamic property, based in part on the likelihood of leader failure—due to mobility, power loss, etc. The large body of existing work in reliability for mobile ad hoc networks certainly should be leveraged.

Support for Disconnected Spheres: Another interesting feature related to reliability that we would like to see developed is support for disconnected spheres. Some spheres are by their nature transient, membership is in flux and what is important is the immediate membership. In other spheres, long-term membership is what is important, and not just the set of members who happen to be connected at any moment. An example of this would be an organizational sphere. A member might wish to send a *DirectedEvent* to other members through their common organizational sphere. In the current Panoply implementation, this would only work if all of the relevant device spheres were concurrently connected to the organizational sphere. If spheres maintained knowledge about their long-term memberships, it would be possible for a sphere to cache events for disconnected members, and then later deliver them when the member reconnects. Panoply already does something similar to this in its handling of vouchers. If a sphere possesses a voucher that is intended for a remote sphere, the system will deliver that voucher encapsulated in an event to the intended recipient when the local sphere is actively connected to the intended sphere. There are clearly issues to deal with in terms of how many events a sphere will cache on behalf of a disconnected child, how long the sphere will cache the events, etc.

9.1.3 Improved Social Context Management

An iterative process of observing, analyzing, and reacting to context is at the heart of Panoply's ability to manage configuration and connections for the mobile user. Currently, we support the discovery and configuration of spheres representing communication networks, locations, and interest-based groups. We also can support social groups through the use of pre-defined rendezvous spheres, such as the team spheres used in the Locative Media Application. Ideally, Panoply would have a richer social model which incorporates the user's social network context to discover and manage the connections to social peers. Unlike organizational groups and physical locations, social networks are user-centered, that is, each user individually has a different social group. These social groups are more difficult to encapsulate within the sphere abstraction; it is not even clear that it is appropriate to do so.

The question of how to incorporate user-centered social structures into Panoply is as of yet unanswered. A recently announced initiative, Open Social [OpenSocial], aims to create an open API allowing access to and interoperation with many existing social networking portals. Presumably, we could also leverage Open Social to access user social networking data, and use this as a discovery mechanism to plug into Panoply. A matching advertisement module could "beacon" relevant data into user social channels. This might include location- or interest-based identifiers, or other pertinent data, and could let users determine when members of their social group were nearby or shared an interest in participating in an activity at a given time.

9.1.4 Improving Coordination Support

The fourth area of improvement for the Panoply infrastructure is in Panoply support for coordination. There are a number of different ways that we can further explore coordination support in Panoply, including looking at generalizing coordination support, as well as improving the ad hoc rescue facility.

General Coordination Support: The existing Panoply implementation provides necessary primitives to enable devices and their applications to discover one another, configure themselves to interoperate, and communicate within their shared context. In the current implementation, with the exception of the rendezvous rescue support, actual coordination must be explicitly implemented at the application layer. We provide several examples of coordination design patterns, notably within the Locative Media Application and Smart Party applications.

In addition to researching other coordination patterns that work well in the Panoply paradigm, we feel that there is room to explore more general coordination support primitives. One possibility is to provide policy primitives for regulating how applications will coordinate within a sphere. Another possibility is to generalize the iterative goal refinement process that we have implemented within the Smart Party. Additionally, the voting mechanism implemented in the Smart Party is another excellent candidate for generalization as a Panoply service. A general voting service could be implemented that can be customized by applications with specific voting algorithms and prioritized or scored application data. Voting-based coordination could then be handled outside of the application, simplifying application development even further.

Improving Ad hoc Rescue: In addition to improving the general coordination support, we can also continue to improve the ad hoc rescue support that Panoply provides. There are two main ways we would like to improve ad hoc rescue support. The first improvement is to change the way ad hoc rescuers deal with connection maintenance. Right now, they break off their existing connections, and must reestablish them after they perform any rescue services. Ideally, we would like those connections to be paused, and then re-connected without requiring additional join overhead. This requires modifying the Sphere Handle object to support a pause in the connection maintenance heartbeats, and then the addition of control messages to initiate this behavior.

The second improvement to ad hoc rescue is better advertisement support. Currently, all ad hoc rescue requesters and rescuers use an identical rendezvous SSID. As a result, rescuers cannot determine the needs of a rescuer a priori, which will result in rescue attempts that will likely fail due to a lack of content or some needed service on the part of the potential rescuer. Further work is necessary to provide more information to would-be rescuers, likely through embedding a task identifier in the rendezvous SSID.

9.1.5 Panoply Support for Resource-Constrained Devices

Another area in need of future development is Panoply's support for resource-constrained devices. We have explored porting Panoply to several resource-constrained devices, including the WRT54G and Nokia 770, with varying degrees of success. These efforts are discussed more in Chapter 4 of this dissertation.

In addition to shrinking Panoply to run with a more limited footprint, we can also consider other ways to support these devices. One possible technique would be to allow resource-constrained devices to be attached to a Panoply sphere as applications. Currently, Panoply's application protocol is based on Java serialization of Panoply events over a loopback network connection. With some small effort, it would be possible to support application communication via a simple message format, possibly XML-based, over an external network connection, such as Bluetooth. Low-powered devices could then interact with a Panoply sphere as Panoply applications. Policy mediation and event subscription management would still occur on the sphere. Effectively, the external device would be using Panoply as an event-based network interface. As the communication channel would be external to the sphere-hosting device, extra care would be needed to secure the channel, using either a pre-shared key or a sideband-based enrollment mechanism, such as the USB-key based location-limited sideband enrollment that we implemented in Panoply.

9.2 Model & Theory

In addition to the infrastructural improvements discussed above, there is also room for future exploration of the general Panoply model. We have considered future work in Sphere design patterns and the dynamic reorganization of spheres, as well as theoretical coordination issues.

9.2.1 Sphere Design Patterns

The existing Panoply-based applications all show a clear benefit from Panoply and the application of the device community paradigm. However, we can classify, into a number of design patterns, the various uses that Spheres of Influence provide within the different application contexts. By formalizing these design patterns, we can begin to analyze application and infrastructure needs and determine when and how to apply each pattern.

Some possible sphere patterns might include:

Transitions/Transformers: The Smart Party demonstrates the transition sphere pattern. The application is built in layers, and each layer acts as a bootstrap for the layer above it. The communication sphere of the party provides localization information, allowing devices to access the location spheres. The location spheres provide a context within which local devices can interoperate. This allows coordinating devices to identify and connect to other interested devices in order to form interest-based spheres.

Amplifiers/Focus Spheres: Interest-based spheres are an example of an amplifier sphere, or focus sphere. A group's goals can be amplified, or focused, through scoped communication. The interest-based spheres in the Smart Party illustrate this pattern—the goal set of the interest-sphere focuses the musical goals of the group. These focused groups are doubly valuable. They serve to both organize and identify members who are striving for a common goal, and also limit coordination traffic to the subset of devices to which it is appropriate and necessary.

Scaling Spheres: This sphere pattern can scale applications by organizing application users and abstracting them into interaction units. This is demonstrated in the Locative Media Application in the form of team spheres, and in the Smart Party in the individual room spheres, and then again in the interest-based spheres.

These patterns clearly need further development, but we believe this is an interesting line of inquiry.

9.2.2 Economics-Based Sphere Reorganization

Another direction we can take Panoply would be to look at dynamic reorganization of spheres—of all types—in response to infrastructure or application needs. For example, if the location infrastructure is experiencing an excess load, it would be in its best interest if the devices in the environment were to aggregate. By aggregating into groups represented by their leaders, the group's impact on the location infrastructure would be reduced. The question is how to encourage devices to aggregate, when it may or may not be in their best interest.

This can be interpreted as a question of economics. The infrastructure can encourage devices to reorganize in a variety of ways by providing a quantifiable reward to accommodating devices. By making the benefit quantitative, devices can automatically analyze the situation and act in the best interest of the user, perhaps augmented by user input. Rewards might take the form of more bandwidth or a slightly larger vote at the Smart Party, even access to some service that non-compliant devices do not receive. Similarly, a greater reward could be given to devices that act as organizers for their peers—sphere leaders in the Panoply model.

Grouping impetus might also come from the user devices themselves, when they recognize the benefit of grouping. The Smart Party exhibits this—infrastructure receives a sizable scaling gain, but user devices also achieve both greater satisfaction and greater fairness for many instances of the Smart Party. We believe that further exploration of dynamic sphere reorganization would be valuable.

9.2.3 A Physics-Based Sphere Model

Another possible model of sphere behavior that we believe would be worth pursuing is one based upon a physical model of attraction and repulsion. The concept is that device spheres are drawn together for various reasons: to participate in a common task, share resources, or access services. Similarly, spheres are also repulsed from one another for various reasons, including incompatible policies, and contrary or competing goals. Additionally, members defect from spheres to join more attractive groups. We see this in the Smart Party when an interest-group member leaves its group to join another group that better fits its goals. It would be interesting to analyze the set of forces that act upon groups of devices that affect their attraction, repulsion, defection, and cooperation.

Research into this model may yield interesting tools for analyzing a user population and predicting various behaviors. This would be useful for infrastructural planning, and for dynamically scaling applications based upon predicted needs.

9.2.4 Exploring Coordination Issues in Pervasive Computing

The third large area of theoretical research that we see involves coordination issues. There is a wide range of behaviors to examine within the context of ubiquitous computing. Our sample applications assume well-meaning collaborators. If collaborators were selfish, or malicious, we could see very different coordination results. It would be useful to examine what happens when individuals with very different motivations collaborate. The ability to detect malicious or selfish behaviors would be extremely valuable in those situations.

There is also interesting existing work that looks at the stability of coordinating groups [Raab2005]. Some of this insight could be applied to group formation rules to help build spheres that are more resilient to defection, and likely to be more stable over time. Other relevant work involves learning better coordination behaviors by observing more successful peers [Hales2006]. The evolutionary approach espoused by Hales et al. could assist underperforming individuals in improving the results of their coordination behavior.

9.3 Large-Scale Developer Evaluation of Panoply Paradigm

In addition to the infrastructure improvements and theoretical exploration we have suggested as future work, there is also the practical issue of evaluating Panoply with a large number of real developers. We have provided anecdotal evidence and some simple complexity measurements indicating that Panoply does simplify development efforts for the applications we have considered. However, as the developers of Panoply, we should reasonably be expected to be able to fully utilize the middleware that we have developed. Ideally, we would perform a more thorough and less biased evaluation of Panoply and Spheres of Influence as a development tool. A full developer evaluation could include a variety of components, including a comparative development study, a reimplementation study, and also a release with developer feedback.

One component of the evaluation would be a study comparing the development efforts of two groups of developers in a controlled environment, such as a computer science class on ubiquitous computing. One group would use Panoply to build a set of applications, while the other, a control group, would build their applications from scratch, using available software development tools. By comparing their experiences, the relative complexity of the applications produced by the groups, and their relative functionality, we would be able to speak more definitively about the benefits of Panoply as a development tool for ubiquitous computing applications.

A second component of this evaluation would be an implementation study, in which a group of developers re-implemented a number of applications from the ubiquitous computing literature, with a focus on how Panoply and Spheres of Influence can improve or extend the original application. Applications considered should include those originally built with toolkits or middleware, as well as those built without. The developer experiences should be documented, and the resulting applications can be compared to the originals in terms of performance, feature set, and software complexity.

Another component of a large-scale evaluation would be a full release of Panoply to the community. This would allow interested individuals to use Panoply to develop their own applications and services. By tracking use and recording developer feedback, we could identify the ways in which Panoply meets developer needs, as well as those in which it falls short. This feedback would also be very useful for identifying new features or services that would make Panoply more useful.

9.4 New Applications

Over the course of developing Panoply and the existing application suite, we have also conceived of a number of other applications that have not yet been implemented due to a lack of time and manpower. This section will briefly summarize several of these applications.

9.4.1 Panoply at the Amusement Park

One application we envisioned is that of an amusement park. A Panoply-based amusement park application could provide a number of interesting services. An obvious service is a mapping and locator service that could help patrons find their away around the park, as well as stay aware of the location of others in their group. This would help prevent family members from getting too lost, and also help groups coordinate their rendezvous. Additionally, such a mapping service could also provide status information on the various rides, including expected wait time and ride restrictions. It could also provide route-planning services to help patrons navigate the park—routes could be planned in such a way that they avoid congested areas of the park.

A second service or application would be meta-entertainment—virtual games or quests that are overlaid upon the normal park attractions. These could include scavenger hunts or other locative entertainment that would take patrons around the park. This type of application could be designed in a similar way to our Locative Media Application.

The application could also be generalized to other settings. For instance, the constrained environment of a cruise ship would similarly benefit from this type of application. A mapping utility would help passengers explore the ship and alert them to relevant events happening in their surroundings. It could also help them find others with common interests, or match up idle passengers who might be interested in getting together for a game.

9.4.2 Office Applications

Within the context of the workplace, there are a number of interesting possible directions for Panoply applications that are aware of location and social contexts. Within our office, we have primarily developed media applications. These applications either automatically route media control events to the appropriate office or dynamically combine preferences from the individuals within a given office for purposes of selecting media to play in that office. In addition to these applications, there are a number of interesting office-related applications.

One example application is a social and location-aware messaging application. This application would allow the user to send an email or message to a group of people dynamically constructed using social and location context. The once fairly static notions of workplace and work hours are being slowly twisted by the emergence of telecommuting, the extended workweek, and alternate work schedules. This makes the once relatively simple task of sending a memo to every member of an organization physically present in the office much more difficult.

Panoply provides membership context and scoping primitives that would simplify the development of such an application. We envision a location-aware instant messenger application that allows the user to construct custom scopes by composing GUI group elements. For example, the user might drag the “LASR” social group tag on top of the “Boelter 3564” location group to create a composed group “LASR members in Boelter 3564.” Messages could then be sent to the newly composed group using existing event scopes.

Another interesting application that we considered would be an application that assists in initiating ad hoc group meetings. Frequently, we wish to have an ad hoc meeting of developers and researchers, but we lack awareness of each other’s presence and schedule. To organize such an ad hoc gathering right now, one goes from door to door, checking to see if the relevant participants are available. A meeting organization application could greatly assist with this process.

An organizing user could formulate the meeting scope in terms of individuals or organization sub-groups who are either necessary or desirable for the meeting. Panoply maintains an awareness of an individual’s presence or absence in the locale; it can alert members to the proposed meeting, and quickly determine, based on user schedule and location, who might possibly attend. Using a to-be-determined value function, Panoply can determine when a satisfactory population is present for the meeting, based upon the meeting scope set by the organizer and the probability of timely user presence in the office. At that point, Panoply can alert the meeting coordinator and attendees, and even select a room for the meeting based on perceived activities in the space.

Chapter 10

Conclusion

The electronic landscape is already quite complex. Users are carrying more and more portable devices, including personal digital assistants, media players, and the ubiquitous mobile telephone. These devices are becoming even more sophisticated, supporting a wide array of wireless technologies and application platforms. User environments are also growing increasingly complicated, equipped with 802.11a/b/g wireless networks, digital video recorders, and a vast array of other networked consumer appliances. Simultaneously, we anticipate an emerging class of applications which integrate user context and preferences with location and social information. Keeping up with these ever-evolving dynamics is a next-to-impossible task for application developers. Worse, keeping up with the changing demands of new and sundry technologies is a frequently frustrating and annoying task for many end users. We see a clear need to provide assistance to these users and application developers. This dissertation presents Panoply, a middleware approach to simplifying device management and application development for ubiquitous computing.

10.1 Contributions of this Dissertation

The contributions of this dissertation are divided into core infrastructure contributions and application contributions. The primary infrastructure contribution is the design, implementation, and evaluation of Panoply, an active middleware system that provides application developers with a general interface to many common communication, configuration, and context services. The underlying implementation of these services is accessed via a general event communication framework; generalizing the interface in this manner allows underlying service implementations to evolve while maintaining a common API.

Additionally, Panoply manages device configuration and connectivity on behalf of the user and their applications, providing flexible discovery, configuration, and connectivity management support. This allows devices to dynamically discover and provision themselves with network configuration data, localization data, environmental policy, and application data. Finally, Panoply actively manages organized communities of devices, referred to as Spheres of Influence. These communities are fundamental building blocks of our ubiquitous computing infrastructure, and are used to structure and manage device communication, system policies, and application interactions. Spheres of Influence provide numerous benefits to users and infrastructure in the form of improved security, availability, scalability, and also application performance.

We evaluated Panoply in several ways. One method of evaluation included measuring a variety of different operating characteristics of Panoply. These measurements indicated that Panoply imposes acceptable overhead and offers reasonable performance in terms of configuration, connection, and communications. In addition to these basic system measurements, Panoply's value has been shown empirically. Together with several student developers, we designed and developed a suite of Panoply applications. These applications demonstrate the utility of Panoply as a development tool. In general, applications were developed quickly. Panoply greatly simplified the development effort by providing easy access to commonly used services and by providing useful abstractions of device relationships and communication.

The second major contribution of this dissertation is in the form of novel applications that make heavy use of Panoply and illustrate the Spheres of Influence paradigm of organized and mediated communities of devices. We applied Panoply and Spheres of Influence to several different and interesting problems, and where possible, have measured different benefits stemming from our approach.

The first application, QED, was technically a service offered through Panoply. QED protects mobile devices and infrastructure by quarantining mobile devices at the point of network ingress and enforcing local policy regarding software updates and system services. To evaluate QED, we analyzed its effectiveness at slowing the spread of an undesirable worm through a mobile population by providing in-situ device quarantine, examination, and decontamination. The results were excellent, indicating that with a 25% level of QED deployment, we could contain the worm spread to 22% of the population that would otherwise be infected. If QED is extended to a 50% level of deployment, then the worm is contained to 2% of the otherwise infected population. These statistics clearly indicate the value of Panoply's mediated join procedure, through which network policy is enforced at the point of ingress. Widespread industry adoption of our model of ingress-based network access control has validated its importance.

Our second featured application is our Locative Media Application (LMA), a team-oriented location-aware media application. The core of the LMA is a group-aware media processor—a Panoply application that manages location and group context in order to customize a media experience for the participants. This application was deployed across the UCLA campus with a non-linear science fiction narrative entitled *nan0sphere*. Teams of participants would experience *nan0sphere,* each as a different character, as they explored various locations on the UCLA campus. One of the issues that the LMA had to deal with on the UCLA campus was the inconsistent availability of wireless connectivity. To address this issue, the LMA provided a mechanism through which devices lacking connectivity and in need of content could advertise their presence. Willing nearby peers formed a transient ad hoc device community in order to rendezvous with and “rescue” the device in need, by providing necessary location-specific content. We evaluated the benefits of this rescue mechanism, and discovered that it was able to significantly improve the availability of the locative media content; additionally, the rendezvous rescue mechanism added only minimal overhead when it was not used.

Our third major application, the Smart Party, highlights a variety of important Panoply features. The actual Smart Party is a dynamic audio experience situated in a set of rooms in some physical environment. As invited guests arrive at the Smart Party environment, their devices, possessing cryptographic party invitations, transparently discover and join the community of devices participating in the Smart Party. Guest devices are securely configured with all necessary network and location information, enabling them to participate fully in the Smart Party application. As guests interact in the Smart Party environment, an audio playlist for each room is dynamically programmed based on the media and preferences of the room’s occupants at that moment. Guest devices combine together into interest-based collaborative groups, allowing them to identify overlapping preferences in a distributed fashion, without relying on centralized coordination or full exposure of user preferences. We have evaluated the Smart Party application as a deployed media experience in the LASR offices in order to measure actual run-time characteristics We have also used simulation to evaluate the performance of our preference coordination mechanism with large parties; this would have been very difficult to realize in practice. Measurements on the deployed Smart Party indicate that users are configured and participating in the Smart Party within ~35 seconds of detecting the environment, well within the bounds of reasonability for the selected application.

Our simulation results indicate that our community-based coordination method significantly out-performs naïve round-robin-based song selection as well as a more sophisticated voting protocol in terms of fairness of song selection. Additionally, our community-based coordination mechanism out-performed or was comparable to the other mechanisms in terms of user satisfaction for nearly all of the Smart Party cases we considered.

Together, these results suggest that the Panoply approach of integrating social constructs into ubiquitous computing applications can significantly improve the user experience. When users are interacting with physically constrained resources, such as wireless connectivity, or preferentially influencing their communal environment, we find that members of Panoply-based device communities can work together to improve communal security, the availability of shared resources, and make better group decisions regarding shared resources.

10.2 The Need for Panoply

Looking forward, we see several different trends in technology and consumer reception of wireless networking technology that point to a need for wider adoption of the principles and techniques that we have explored with Panoply. User frustration with the complexities of home networking and configuration has proven to be a major headache for manufacturers, and a general impediment to the adoption of new technologies. With return rates between 20-30%, home-networking hardware has recently been the most returned item at the major consumer electronics retailers [Scherf2002]. As recently as 2006, more than a quarter of purchased wireless access points were returned [MacMillan2006], due, not to hardware failures, but instead to the inability of consumers to integrate the devices into their home networks. This frustration with the ease of use and configuration is frequently cited as a major problem with wireless networking [Laszlo2002]. It is important to note that these reported frustrations and problems revolve around network and application configuration issues within consumer's *own home*, an environment in which the user typically has a great deal of control and possibly time with which to set up his devices.

Users engaging in many pervasive computing applications will, however, interact in unplanned ways in foreign environments, often with mutually unknown entities [Ramakrisha2007-2]. This will exacerbate these frustrations, as users will need to discover one another and configure their devices to interact in an on-demand manner, and in environments that are outside of user control. Emerging wireless applications, such as the music-sharing facility provided by the Microsoft Zune, are already heading in this direction, and indeed, we see that these early efforts have exhibited a number of usability problems [Dilger2007].

Additionally, in the coming years, many different types of user devices will be interacting with more and more wireless networks in the workplace, public places, and in the user and friends' homes. Classes of mobile devices that have traditionally used only closed networks, such as cellular telephones, or lacked networking facilities entirely, such as personal media players, are now being equipped with integrated 802.11 radios. We believe that an increasing number of users will attempt to use this new functionality in the places they visit and that novel applications will emerge to leverage this new connectivity. This has two important implications. First, if users are to make effective use of this new connectivity, their devices must be able to easily detect, configure, and integrate themselves into newly encountered wireless networks. Second, these networks must be able to effectively protect themselves and apply local policy to network admission and application interactions.

We believe that the automatic, application-driven configuration demonstrated by Panoply is one way to simplify the configuration issues that have frustrated many users. Additionally, Panoply's mediated access model provides a solid basis for network admission and policy-driven application-level security that will be critical as user devices interact with unknown devices and in foreign environments.

Finally, we expect to see more and more users interacting within physical locations with their mobile devices, both with one another, and with location-specific services. As noted above, we expect to see new applications that utilize the emerging communication capabilities of many different types of mobile devices, enabling them to interact with the surrounding environment and user social networks, as well as with other nearby devices. Some applications are already emerging in this space; Yahoo's recent announcement of a location and social-network event planning application [Reedy2008] is one example, albeit offered in a closed and provider-controlled manner. Others, including Miklas et al. [Miklas2007], have also shown the benefits of extending social awareness to mobile systems for a variety of applications. As other new applications are conceived of and developed, we anticipate that it will be desirable for users to coordinate their behaviors, particularly when interacting under physical constraints, such as location, or with shared environmental phenomena, e.g. temperature, lighting, projected video, shared audio, etc. We believe the Panoply approach is a good step in the direction of a general-purpose architecture for coordinating user interactions within these types of applications. In addition to simplifying and streamlining device configuration, we provide an example of this with our Smart Party, which demonstrates coordination and collaborative behavior among user devices. We believe that this type of organization will be necessary as consumer devices become more aware of their environment and participate in social- and location-aware applications.

10.3 Connecting Panoply to Human Needs

Panoply has successfully enabled developers to quickly design, implement, and deploy interesting and complex applications that leverage wireless networks, location context, user social context, and also other nearby devices. This is accomplished through an iterative configuration and interaction process in which devices seek out secure communication channels, and then extend and refine their interactions through identifying and acquiring additional layers of context, including application context, physical location context, and user social context.

This model of device configuration feels natural, and in many ways mirrors the division and classification of human needs set out by Abraham Maslow [Maslow1943.] Maslow's *Hierarchy of Needs*, is typically depicted as a pyramid consisting of five levels, each consisting of a different class of need. Figure 35 depicts one version of Maslow's hierarchy. The lower four levels of the hierarchy are regarded as physiological needs, including basic physical needs, safety needs, the need for belonging, and also esteem needs. The top tier of the needs pyramid represents human self-actualization, which includes needs for self-fulfillment, morality, creativity, etc. Maslow's hierarchy is roughly an ordering of the importance of the different basic needs, or the order in which human needs must typically be satisfied.

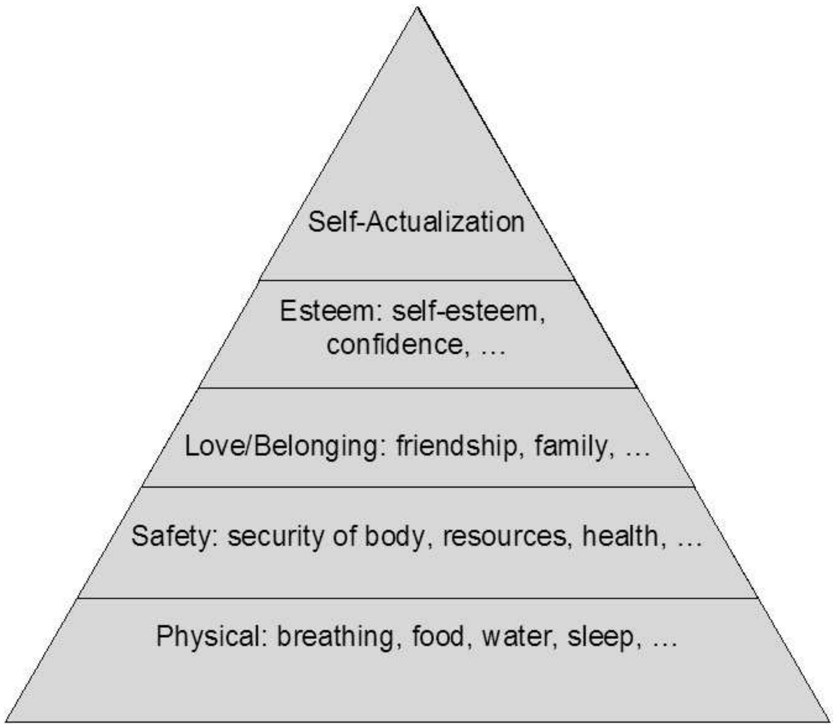

Self-Actualization
Esteem: self-esteem, confidence, ...
Love/Belonging: friendship, family, ...
Safety: security of body, resources, health, ...
Physical: breathing, food, water, sleep, ...

Figure 35. Abraham Maslow's *Hierarchy of Needs*

Below, in Figure 36, we see the Panoply device requirement hierarchy. This pyramid illustrates the basic stages of the Panoply device configuration process, from acquiring connectivity to actively representing the human user in social- and location-based applications. The Panoply device requirements hierarchy greatly resembles Maslow's hierarchy in its structure and organization. The more fundamental the requirement, the lower the level at which you find it in the hierarchy.

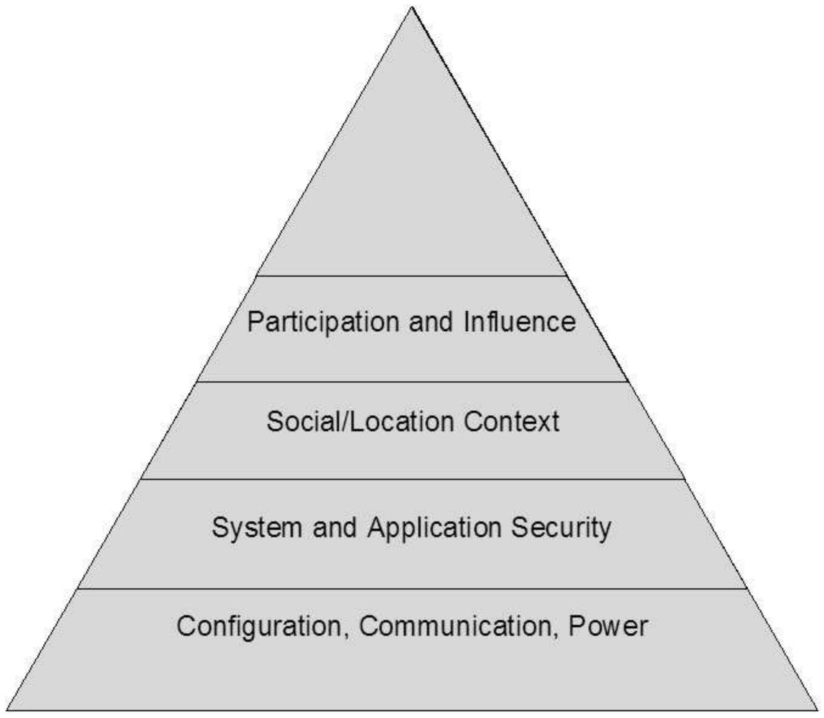

Participation and Influence
Social/Location Context
System and Application Security
Configuration, Communication, Power

Figure 36. Panoply hierarchy of device requirements

10.3.1 Physical Needs and Security

The lowest level of the Panoply hierarchy consists of physical device needs, including communication, and configuration. These are the first requirements Panoply attempts to satisfy, through discovery and sphere vouchers. These correspond closely to physiological needs in Maslow's hierarchy, of food, water, sleep, etc. Without these, the human ceases to function; similarly, without power or a usable network configuration, a mobile device is of limited use. The second device requirement tier consists of device security needs; this tier corresponds well to the safety needs tier in Maslow's own hierarchy. Maslow's safety tier includes the need for health, protection from harm, and financial security. In the Panoply requirement hierarchy, the security tier includes the secure setup of a Panoply sphere connection, possibly using an enrollment mechanism, subject to the policy of both the device and the sphere to which the device connects.

10.3.2 Community Membership and Social Acknowledgement

The third tier of Maslow's hierarchy includes the human needs for belonging, family and love. These needs are all community- or group-oriented. The third tier of the Panoply device requirements includes social and location context. Similar to the corresponding tier of Maslow's hierarchy, these requirements describe the device's need to participate in, or "belong," to the appropriate device community. Social and location context allow the device to identify and join with the appropriate devices in order to participate in application-based, social-based, and location-based device communities.

The fourth tier of Maslow's hierarchy, the esteem needs tier, builds on the community aspects of the love and belonging tier. The esteem needs tier represents the individual's need to be respected and acknowledged within his or her social circles. In Panoply, after devices have joined with location- or social-based device communities, they typically participate in applications or express preferences with respect to some shared environmental resource. This participation and influence tier is very similar in that regard to the esteem needs tier of Maslow's hierarchy.

We believe that these similarities are due to more than coincidence or intentional design. Panoply's hierarchy of device requirements describes the functionality that Panoply developers require in order to create useful tools and applications that users can rely on. Ultimately, the mobile device is an extension of the human in the virtual realm. Panoply is reinterpreting human needs through the mobile device, and it makes sense that we see

Maslow's hierarchy of needs reinterpreted in the device domain. One question remains, however: what is at the top of the device requirement hierarchy? Maslow places *self-actualization* at the top of his hierarchy of human needs. Self-actualization refers to the instinctual human need to make the most of one's abilities. We can conjecture that in the pervasive computing arena, the equivalent requirement for devices might be to correctly anticipate users' needs and desires, requiring only minimal user input. Alternately, perhaps our devices might help us to better achieve our own self-actualization goals.

The lower four tiers of Maslow's hierarchy are also known as the *d-needs*, or *deficiency needs*, as the individual is not typically happy or satisfied solely through meeting those needs; however, they are anxious and dissatisfied if these needs are not met. The individual can achieve the most happiness and satisfaction by satisfying needs in the self-actualization tier. Our breakdown of Panoply's device requirements can be also subjected to a similar analysis. If we consider our device to be our extension into smart spaces and intelligent locations, an application or device that does not satisfy the lower four tiers of the Panoply requirements is inadequate. It would not suitable for common pervasive computing scenarios if the fundamental requirements for representation and influence cannot be met. However, simply meeting those needs is not sufficient to actualize the so-called invisible computing paradigm. We believe that this can be accomplished only by reaching beyond the representational needs to the elusive top tier of the device requirement hierarchy. Whether that is through predictive and automated decision-making, or by otherwise improving the human experience through technology mediation, Panoply provides a solid base upon which to build those applications.

10.4 Final Remarks

This dissertation has presented Panoply, active middleware for pervasive computing applications. Panoply provides the developer with the tools and framework necessary to support many different types of pervasive computing applications, including basic wireless configuration and communication support, security primitives, discovery support for location and social context, and device grouping facilities to support community participation and collaboration. Panoply provides the end user with an active device management system that handles complex configuration and context acquisition tasks on behalf of the user, with minimal user intervention. We have evaluated Panoply in the context of three different user application scenarios, and have shown both that Panoply simplifies the development process of pervasive computing application, and also that we can realize substantial benefits from Panoply-enabled applications.

TRADEMARKS

Appendix I. Panoply Vouchers

The Panoply Voucher data structure was developed jointly with Jason Karuza, and is described fully in [Karuza05]. Figure 37 details the internal structure of a single voucher. The various components of the voucher data structure are briefly described below.

```
                        Voucher Structure
0                   1                   2                   3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|              Type             |            Version            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|        Voucher_ID_Size        |        Vouchee_ID_Size        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                      Start_Date_And_Time  ...                 |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                 ...  Start_Date_And_Time                      |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                       End_Date_And_Time  ...                  |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                  ...  End_Date_And_Time                       |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                       Opaque_Field_Size                       |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                          Voucher_ID                           |
|                               .                               |
|                               .                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                          Vouchee_ID                           |
|                               .                               |
|                               .                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                    Opaque_Field (optional)                    |
|                               .                               |
|                               .                               |
|                               .                               |
|                               .                               |
|                               .                               |
|                               .                               |
|                               .                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 37. The Panoply voucher data structure

Type Field: The type field is used to describe the purpose of the voucher. As of February 2008, we currently support three types of vouchers, the social voucher, the location voucher and a generic voucher. Other voucher types can be added by extending the VoucherOptions structure.

Version Field: The version field indicates which version of the voucher data structure is used by the voucher in question. This allows us to deal with changes in voucher structure over time.

Voucher ID Size: The length of the voucher ID field. This allows for ID encodings of differing lengths.

Vouchee ID Size: The length of the vouchee ID field.

Start Date and Time: The time at which a voucher becomes active, specified as the number of milliseconds that have elapsed since January 1, 1970, 00:00:00 GMT.

End Data and Time: The time at which a voucher expires, specified as the number of milliseconds that have elapsed since January 1, 1970, 00:00:00 GMT.

Opaque Field Size: The length of the opaque field. This allows for arbitrary extensions to be included in the opaque data.

Voucher ID: The encoded identity of the entity that generated the voucher.

Vouchee ID: The encoded identity of the entity for whom the voucher is intended.

Opaque Field: The opaque field is a binary blob that is intended for extensions or for specifying data needed by different voucher types. For instance, vouchers to be used for voting might specify voting constraints, or viable proxies. Location vouchers would include MAC/SSID trigger information, as well as a network configuration object.

Signature: The entity generating the voucher cryptographically signs the voucher using DSA with SHA-1.

Appendix II. Panoply Event Types

APPLICATION: This is the master event type for all application events.

COMPONENT: Events of this type are used to launch a Panoply component or application inside the context of a running Sphere.

CONNECTIVITY: Connectivity events communicate changes in the state of the sphere's network connectivity.

CREDENTIAL: These are used to issue or verify vouchers.

DIRECTED: This event type is used for source routing of events.

DOORMAN: Events of this type communicate requests or updates from the external sphere interface.

EVENT: This event type is used for communication with the event processor, event subscriptions, unsubscriptions, etc.

EXTERNAL: This is used for external observations, discovery announcements, etc.

HEARTBEAT: Heartbeat events are used as in-band keep-alive messages between active sphere relationships.

LOCATION: This event type is used for location updates or refresh events.

MEMBERSHIP: Events of this event type are used to exchange information about changes in sphere membership status—when we join a new sphere or a new sibling sphere comes into our sphere, etc.

POLICY: This event type is used for policy messages.

SPHEREMANAGEMENT: This type is used for events that manage relationships among interest-based spheres, exchange leadership proposals, etc.

STATE: This event type is used to access the sphere state cache—either to check or update a (key,value) pair.

TIMEWARP: Timewarp events are used to alert components that a significant time discrepancy has been detected.

References

[Adjie-Winoto1999] Adjie-Winoto, W., Schwartz, E., Balakrishnan, H. Lilley, J. “The design and implementation of an intentional naming system.” ACM SIGOPS Operating Systems Review, v.33 n.5, p.186-201, December 1999.

[Al-Muhtadi2003] Al-Muhtadi, J., Ranganathan, A., Campbell, R. and Mickunas, M. “Cerberus: A Context-Aware Security Scheme for Smart Spaces.” *Proceedings of the 1st IEEE International Conference on Pervasive Computing and Communications* (PerCom'03), March 23-26, 2003.

[Al-Muhtadi2004] Al-Muhtadi, J., Chetan, S., Ranganathan, A. and Campbell, R. "Super Spaces: A Middleware for Large-Scale Pervasive Computing Environments." *Proceedings of Perware '04: IEEE International Workshop on Pervasive Computing and Communications*, Orlando, Florida, March 2004.

[Anderson2005] Anderson, E., Eustice, K. Markstrum, S., Hansen, M., and Reiher, P. “Mobile Contagion: Simulation of Infection and Disease.” *Proceedings of the 2005 Symposium on Measurement, Modeling and Simulation of Malware*, June 2005.

[Apple] Apple Rendezvous http://www.apple.com/macosx/features/rendezvous/

[Audioscrobbler] Audioscrobbler: The Social Music Technology Playground http://www.audioscrobbler.net/ Retrieved on November 29, 2007.

[Bahl2000] Bahl, P. and Padmanabhan V. N., Radar: An In-Building User Location and Tracking System, *Proceedings of IEEE Conference on Computer Communication (Infocom '00),* Vol. 2, pp. 775-784, March 2000.

[Balfanz2004] Balfanz, D., Durfee, G., Grinter, R., Smetters, D., and Stewart, P. “Network-in-a-Box: How to Set Up a Secure Wireless Network in Under a Minute.” *Proceedings of USENIX Security 2004.*

[Bardram2004] Bardram, J., and Hansen, T. “The AWARE Architecture: Supporting Context Mediated Social Awareness in Mobile Cooperation.” *Proceedings of the 2004 ACM Conference on Computer Supported Cooperative Work* (CSCW2004).

[Barrenho2006] Barrenho, F., Romao, T., Martins, T, and Correia, N. “InAuthoring environment: interfaces for creating spatial stories and gaming activities.” In *Proceedings of the 2006 ACM SIGCHI Intl. conference on Advances in Computer Entertainment Technology*, Hollywood, CA.

[Bjork2001] Bjork, S., Falk, J., Hansson, R., and Ljungstrand, P. "Pirates! - Using the Physical World as a Game Board." In *Proceedings of Interact 2001*. Tokyo, JAPAN. (July 2001)

[Blaster] CERT Advisory CA-2003-20 W32/Blaster worm http://www.cert.org/advisories/CA-2003-20.html Retrieved November 24, 2007.

[Borriello2005] Borriello G., Chalmers M., LaMarca A., Nixon, P.: Delivering Real-World Ubiquitous Location Systems. Communications of ACM, Vol. 48, No. 3. (March 2005) 36—41

[BitTorrent] http://www.bittorrent.com Retrieved April 2007.

[Bottazi2003] Bottazzi, D. Corradi, A. Montanari, R. "AGAPE: a location-aware group membership middleware for pervasive computing environments." Proceedings of the 8th IEEE International Symposium on Computers and Communication, 2003. (ISCC 2003).

[BouncyCastle] The Legion of the Bouncy Castle. http://www.bouncycastle.org/ Retrieved September 26, 2007.

[Brams2002] Brams, Steven J. & Fishburn, Peter C., 2002. "Voting procedures" in Handbook of Social Choice and Welfare. Ed. K. J. Arrow & A. K. Sen & K. Suzumura (ed.), Elsevier Press. 2002. p. 173-236

[Brooks1997] Brooks, R. "The Intelligent Room Project." *Proceedings of the 2nd Intl. Cognitive Technology Conference.* 1997. Aizu, Japan.

[Capkun2001] Capkun S., Hamdi M., and Hubaux J., "GPS-Free Positioning in Mobile Ad Hoc Networks," *Proceedings of Hawaii International Conference on System Science,* January 2001.

[Chetan2004] Shiva Chetan, Jalal Al-Muhtadi, Roy Campbell and M.Dennis Mickunas, "A Middleware for Enabling Personal Ubiquitous Spaces," UbiSys '04: System Support for Ubiquitous Computing Workshop at Sixth Annual Conference on Ubiquitous Computing (UbiComp 2004), Nottingham, England, Sept. 2004.

[Chou2005] Chou, S.-C., Hsieh, W.-T., Gandon, F., and Sadeh, N.: "Semantic Web Technologies for Context-Aware Museum Tour Guide Applications," *Proceedings of WAMIS 2005.* (March 2005)

[Cisco2003] White Paper - Cisco NAC: The Development of the Self-Defending Network.http://www.cisco.com/en/US/netsol/ns340/ns394/ns171/ns413/networking_solutions_white_paper09186a00801e0032.shtml

[Classpath] GNU Classpath – an open-source implementation of the core Java libraries http://www.gnu.org/software/classpath/

[Coen1999] M. Coen, B. Phillips, N. Warshawsky, L. Weisman, S. Peters, P. Finin, "Meeting the Computational Needs of Intelligent Environments: The Metaglue System," *Proceedings of International Workshop on Managing Interactions in Smart Environments* (MANSE '99), 1999.

[Corner2002] Corner, M. and Noble, B. "Zero-Interaction Authentication," *Proceedings of Mobicom 2002.*

[Crawdad] Crawdad: A Community Resource for Archiving Wireless Data At Dartmouth. http://crawdad.cs.dartmouth.edu/ Retrieved on November 24, 2007.

[Cugola2001] Cugola, G. and Di Nitto, E. "Using a publish/subscribe middleware to support mobile computing." *Proceedings of the Workshop on Middleware for Mobile Computing,* Heidelberg, Germany, Nov. 2001.

[Dey2006] Dey, A. K., Sohn T., Streng S., and Kodama J., "iCAP: Interactive Prototyping of Context-Aware Applications," *Proc. Fourth Intl. Conference on Pervasive Computing*, 2006

[Dilger2007] Dilger, Daniel "Why Microsoft's Zune is Still Failing." November 2007. http://www.roughlydrafted.com/2007/11/22/why-microsofts-zune-is-still-failing Retrieved on January 3, 2008.

[Dodgeball] Dodgeball mobile social software. http://www.dodgeball.com Retrieved April 2007.

[eDocent] eDocent Website http://www.ammi.org/site/extrapages/edoctext.html

[Eisenstadt2002] Eisenstadt, M. and Dzbor, M. "BuddySpace: Enhanced Presence Management for Collaborative Learning, Working, Gaming and Beyond," *Proceedings of JabberConf Europe 2002.*

[Eustice1999] Eustice, K., Lehman, T., Morales, A., Munson, M., Edlund, S., and Guillen, M. "A Universal Information Appliance," IBM Systems Journal 38(4): 575-601 1999.

[Eustice2003a] Eustice, K., Kleinrock, L., Markstrum, S., Popek, G., Ramakrishna, V. and Reiher, P. "Enabling Secure Ubiquitous Interactions," *Proceedings of the International Workshop on Middleware for Pervasive and Ad-hoc Computing* (MPAC) held in conjunction with Middleware 2003.

[Eustice2003b] Eustice, K., Kleinrock, L., Markstrum, S., Popek, G., Ramakrishna, V. and Reiher, P. "Securing WiFi Nomads: The Case for Quarantine, Examination, and Decontamination," *Proceedings of the New Security Paradigms Workshop* (NSPW) 2003.

[Fiege2000] Fiege, L., and Mühl, G. "Rebeca Event-Based Electronic Commerce Architecture", 2000. http://www.gkec.informatik.tu-darmstadt.de/rebeca.

[Fiege2002] Fiege, L., Mezini, M., Mühl, G., and Buchmann, A. "Engineering Event-Based Systems with Scopes," *Proceedings of ECOOP '02.*

[Garcia-Molina1982] Garcia-Molina, H. "Elections in a Distributed Computing System," IEEE Trans. on Computers, C-31(1):48-59, Jan. 1982. 11

[Gajos2001] Gajos, K., Weisman, L. and Shrobe, H. "Design principles for resource management systems for intelligent spaces," *Proceedings of The Second International Workshop on Self-Adaptive Software*, Budapest, Hungary, 2001.

[Gini1912] Gini C. "Variabilitá e mutabilita" 1912 reprinted in *Memorie di metodologica statistica* (Ed. Pizetti E, Salvemini, T). Rome: Libreria Eredi Virgilio Veschi 1955.

[Glazer2007] Glazer, G. "Time Space Limited Sideband Enrollment." Masters Comprehensive, UCLA Computer Science 2007.

[Goodrich2005] Goodrich, M., Sirivianos, M., Solis, J., Tsudik, G., and Uzun, E. "Loud and Clear: Human-Verifiable Authentication Based on Audio," *Proceedings of the 26th IEEE International Conference on Distributed Computing System.s*

[Hales2006] Hales, David, and Arteconi, Stefano, "SLACER: A Self-Organizing Protocol for Coordination in Peer-to-Peer Networks," IEEE Intelligent Systems v.21 n.2 (March 2006) pp. 29-35.

[Hanssens2002] Hanssens, N., Kulkarni, A., Tuchinda, R. and Horton, T. "Building Agent-Based Intelligent Workspaces." *Proceedings of the ABA.* June 2002.

[Harrison1996] Harrison, S. and Dourish, P. "Re-place-ing space: the roles of place and space in collaborative systems," *Proceedings of the 1996 ACM Conference on Computer Supported Cooperative Work* (CSCW 1996).

[Hilley2003] Sarah Hilley. "MSBlaster could have been MUCH worse," CompSec Online, Aug 13, 2003. http://www.compseconline.com/analysis/ 030813msblaster.html

[Hindmarsh2002] Hindmarsh, J., Heath, C., vom Lehn, D., Cleverly, J.: “Creating Assemblies: Aboard the Ghost Ship,” *Proc. 2002 ACM Conference on Computer Supported Cooperative Work.*

[Hoebeke2006] Hoebeke, J., Holderbeke, G. , Moerman, I. Jacobsson, M., Prasad, V., Cempaka Wangi, N., Niemegeers, I., Heemstra de Groot, S. Personal network federations In the Proceedings of the 15th IST Mobile and Wireless Communications Summit 2006.

[Hull2004] Hull, R., Clayton, B., Melamed, T.: “Rapid Authoring of Mediascapes,” In *Proceedings of Ubicomp 2004*.

[Iptables] netfilter/iptables project homepage http://www.netfilter.org/ Retrieved on November 24, 2007.

[Iwasaki2003] Iwasaki, Y., Kawaguchi, N. and Inagaki, Y. “Touch-and-Connect: A Connection Request Framework for Ad-Hoc Networks and the Pervasive Computing Environment,” Proceedings of the First IEEE International Conference on Pervasive Computing and Communications (PerCom'03), March 23-26, 2003.

[Jacobsen2001] Jacobsen, H. A. “Middleware Services for Selective and Location-based Information Dissemination in Mobile Wireless Networks,” *Proceedings of the Workshop on Middleware for Mobile Computing,* Heidelberg, Germany, Nov. 2001.

[Jacobsson2005] Jacobsson, M. et al. “A Network Architecture for Personal Networks,” Proceedings of the 14th IST Mobile & Wireless Communications Summit, Dresden, Germany, Jun 19-22, 2005.

[Jazelle] Jazelle Execution Environment Acceleration. http://www.arm.com/products/esd/jazelle_home.html Retrieved on September 26, 2007.

[Jessie] Jessie: A Free Implementation of the JSSE. http://www.nongnu.org/jessie/ Retrieved September 26, 2007.

[Johnson1996] Johnson, D. and Maltz, D. “Dynamic source routing in ad-hoc wireless networks.” *Proceedings of SIGCOMM '96.*

[Jpcap] Jpcap -- a network packet capture library http://jpcap.sourceforge.net/ Retrieved on November 24, 2007.

[Jung2007] Jung, S., Lee, U., Chang, A., Cho, D. and Gerla, M. "BlueTorrent: Cooperative Content Sharing for Bluetooth Users," *Proceedings of IEEE PerCom'07.*

[Kaffe] Kaffe Java Virtual Machine http://www.kaffe.org/ Retrieved on October 6, 2007.

[Kagal2001a] Kagal, L., Korolev, V., Chen, H., Joshi, A., and Finin, T., "Centaurus: A Framework for Intelligent Services in a Mobile Environment", 21st International Conference on Distributed Computing Systems Workshops (ICDCSW '01), April 16 - 19, 2001, Mesa, Arizona.

[Kagal2002b] Kagal, L., Undercoffer, J., Perich, F., Joshi, A., and Finin, T. "A Security Architecture Based on Trust Management for Pervasive Computing Systems," In Proceedings of Grace Hopper Celebration of Women in Computing 2002.

[Kahney03] Kahney, L. "Feel Free to Jack Into My IPod." *Wired Magazine.* November 21, 2003.

[Karuza2005] Karuza, Jason. "Enrollment Vouchers." Masters Comprehensive, UCLA Computer Science 2005.

[Kidd1999] C. D. Kidd, R. J. Orr, G. D. Abowd, C. G. Atkeson, I. A. Essa, B. MacIntyre, E. Mynatt, T. E. Starner and W. Newstetter, "The Aware Home: A Living Laboratory for Ubiquitous Computing Research," In the Proceedings of the Second International Workshop on Cooperative Buildings - CoBuild'99, October 1999.

[Kindberg2000] Kindberg, T. and Barton, J. "A web-based nomadic computing system." *Computer Networks,* 1999.

[Kindberg2002] Kindberg, T. et. al. "People, Places, Things: web presence for the real world." *Mobile Networks and Applications.* Vol 7, issue 5. October 2002.

[Kindberg2003] Kindberg, T. Zhang, K. "Secure Spontaneous Device Association," *In the proceedings of the Fifth International Conference on Ubiquitous Computing*, October 12-15, 2003.

[King2007] King, T., Butter, T., Brantner, M., Kopf, S., Haenselmann, T., Biskop, A., *Färber, A., Effelsberg, W.* "Distribution of Fingerprints for 802.11-based Positioning Systems." *Proceedings of the 8th International Conference on Mobile Data Management* (MDM 2007), pp. 224-226, Mannheim, Germany, 10. May 2007.

[Kleinrock1996] Kleinrock, L., "Nomadicity: Anytime, Anywhere In A Disconnected World", *Mobile Networks and Applications.* Vol. 1, No. 4, January 1996, pp. 351-357.

[Kleinrock2001] Kleinrock, L. "Breaking Loose.", *Communications of the ACM*, Vol 44, No. 9, pp 41.45, September 2001.

[Koile2003] Koile, K., Tollmar, K., Demirdjian, D., Shrobe, H. and Darrell, T. “Activity Zones for Context-Aware Computing,” *In the Proceedings of the Fifth International Conference on Ubiquitous Computing*, October 12-15, 2003.

[Kottahachchi2004] Kottahachchi, B. and Laddaga, R. “Building Access Controls for Intelligent Environments,” *Proceedings of ISDA '04: 4th Annual International Conference on Intelligent Systems Design and Applications.* Budapest, Hungary, August 2004.

[Kwak2004] Kwak, S.Y.: “Designing a Handheld Interactive Scavenger Hunt Game to Enhance Museum Experience,” *MA Thesis*. Michigan State University. (2004)

[LaMarca2005]. LaMarca, A. et al. “Place Lab: Device Positioning Using Radio Beacons in the Wild,” *Proceedings of Pervasive 2005*, Munich, Germany, May 2005.

[Laszlo2002] Laszlo, J. Home Networking: Seizing Near-Term Opportunities to Extend Connectivity to Every Room. Jupiter Research, July 2002.

[Levin1995] Levin, Jonathan and Nalebuff, Barry. “An Introduction to Vote-Counting Schemes.” In the *Journal of Economic Perspectives.* v.9 no. 1. Winter 1995, pp. 3-26.

[Libpcap] Tcpdump/libpcap http://www.tcpdump.org/ Retrieved on November 24, 2007.

[MacMillan2006] MacMillan, R. “Wireless networking baffles some customers.” Reuters News Report, March 2006.

[MAGNET] MAGNET Beyond http://www.ist-magnet.org Retrieved October 30, 2007.

[Maslow1943] Maslow, Abraham. *A Theory of Human Motivation*, Psychological Review 50 (1943):370-96.

[McCarthy1998] McCarthy, J. and Anagost, T. “MusicFX: An Arbiter of Group Preferences for Computer Supported Collaborative Workouts,” *Proceedings of the ACM 1998 Conference on Computer Supported Cooperative Work.*

[McCune2005] McCune, J.M., Perrig, A., and Reiter, M.K. “Seeing is Believing: Using Camera Phones for Human-Verifiable Authentication.” *Proceedings of the 2005 IEEE Symposium on Security and Privacy,* Oakland, California, pp 110–124.

[mDNS] Multicast Domain Name Service. http://www.watersprings.org/pub/id/draft-manning-dnsext-mdns-00.txt Retrieved October 17, 2007.

[Miklas2007] Miklas, A. et al. ”Exploiting Social Interactions in Mobile Systems,” *Proceedings of the 9th International Conference on Ubiquitous Computing (UbiComp)*, Innsbruck, Austria, September 2007.

[MSWZC] Microsoft Wireless Zero Configuration Reference. http://msdn2.microsoft.com/en-us/library/ms706593.aspx Retrieved on November 8, 2007.

[NAC] Network Access Control entry on Wikipedia. http://en.wikipedia.org/wiki/Network_Access_Control Retrieved on November 1, 2007.

[Necula1997] Necula, George C. “Proof-carrying code.” *Proceedings of the 24th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL97).*

[Nikolova2007] Nikolova, E., Tamari, H., Saha, A., and Schrader, A. ”Pervasive Campus: Smart Party,” *Proceedings of the 2nd International Conference of Digital Live Art 2007.*

[Nmap] Nmap – Free Security Scanner for Network Exploration & Security Audits http://insecure.org/nmap/ Retrieved on November 24, 2007.

[O’Hara2004] O’Hara, K., Lipson, M., Jansen, M., Unger, A., Jeffries, H., and Macer, P. “Jukola: Democratic Music Choice in a Public Space.” *Proceedings of the 2004 Conference on Designing Interactive Systems: Processes, Practices, Methods, and Techniques.*

[OMG] Object Management Group. http://www.omg.org. Retrieved April 2007.

[OpenSocial] Open Social http://code.google.com/apis/opensocial/ Retrieved on November 20, 2007.

[Peters2003] Peters, S. et al. “Hyperglue: Designing High-Level Agent Communication for Distributed Applications,” Originally submitted to AAMAS'03.

[Pogue2007] Pogue, D. “The Trouble with Home Networking.” The New York Times, April 12, 2007. http://pogue.blogs.nytimes.com/2007/04/12/the-trouble-with-home-networking/

[Prasad1999] Prasad, M. and McCarthy, J. ”A Multi-Agent System for Meting Out Influence in an Intelligent Environment” *Proceedings of the Eleventh Innovative Applications of Artificial Intelligence Conference (IAAI '99).*

[Quick2004] Quick, K., Vogiazou, Y. “CitiTag Multiplayer Infrastructure,” *TR: KMI-04-7* (March 2004).

[Raab2005] Raab, Philippe. "Can Endogenous Group Formation Prevent Coordination Failure? A Theoretical and Experimental Investigation" (June 2005). IZA Discussion Paper No. 1628. Available at SSRN: http://ssrn.com/abstract=738364

[Ramakrishna2007] Ramakrishna, V., Eustice, K., and Reiher, P. "Negotiating Agreements Using Policies in Ubiquitous Computing Scenarios," *Proceedings of the 2007 IEEE International Conference on Service-Oriented Computing and Applications (SOCA 2007).*

[Ramakrishna2007-2] Ramakrishna, V., Eustice, K., and Schnaider, M. "Approaches for Ensuring Security and Privacy in Unplanned Ubiquitous Computing Interactions." *Mobile and Wireless Network Security and Privacy.* Ed. Makki et al. 2007.

[Reedy2008] Reedy, Sarah. CES: Yahoo delivers on wow factor. Telephony Online, January 8, 2008. http://blog.telephonyonline.com/unfiltered/2008/01/08/ces-yahoo-delivers-on-wow-factor/ Retrieved on January 10, 2008.

[Román2002] Román, M., Hess, C., Cerqueira, R., Ranganathan, A., Campbell, R. and Nahrstedt, K. Gaia: A Middleware Infrastructure to Enable Active Spaces. IEEE Pervasive Computing, pp. 74-83, Oct-Dec 2002.

[RFC1305] Mills, David. RFC 1305 "Network Time Protocol v3." http://www.ietf.org/rfc/rfc1305.txt Retrieved October 4, 2007.

[RFC2131] Droms, R. RFC 2131 "Dynamic Host Configuration Protocol." http://www.ietf.org/rfc/rfc2131.txt Retrieved on November 5, 2007.

[RFC2132] Alexander, S. and Droms, R. RFC2132 "DHCP Options and BOOTP Vendor Extensions." http://www.ietf.org/rfc/rfc2132.txt Retrieved on November 24, 2007.

[RFC2865] Rigney, C. et al. RFC 2865 "Remote Authentication Dial-In User Service" http://www.ietf.org/rfc/rfc2865.txt Retrieved on November 5, 2007.

[RFC3118] Droms, R. and Arbaugh, W. RFC3118 "Authentication for DHCP Messages" http://www.ietf.org/rfc/rfc3118.txt Retrieved on November 24, 2007.

[RFC3748] Aboba, B. et al. RFC 3748 "Extensible Authentication Protocol." http://www.ietf.org/rfc/rfc3748.txt Retrieved on November 5, 2007.

[RFC3927] Cheshire, S., Aboba, B., and Guttman, E. RFC3927 "Dynamic Configuration of IPv4 Link-Local Addresses" http://www.ietf.org/rfc/rfc3927.txt Retrieved on November 25, 2007.

[RXTX] RXTX: The Prescription for Transmission http://users.frii.com/jarvi/rxtx/ Retrieved on November 23, 2007.

[SableVM] SableVM Project http://www.sablevm.org/ Retrieved October 6, 2007.

[Scherf2003] Scherf, K. Panel on home networking. Consumer Electronics Associate (CEA) Conference, 2003.

[Schmalstieg2005] Schmalstieg, D. and Wagner, D.: "A Handheld Augmented Reality Museum Guide," In *Proc. IADIS Intl Conf. on Mobile Learning (ML2005)*. Qawra, Malta. (June 2005)

[Sekar2003] Sekar, R., Venkatakrishnan, V., Basu, S., Bhatkar, S and DuVarney, D. "Model-carrying code: a practical approach for safe execution of untrusted applications," *In the Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles*, October 19 - 22, 2003.

[Smith2005] Smith, I., et al. "Social Disclosure Of Place: From Location Technology to Communication Practice," *Proceedings of Pervasive 2005*, Munich, Germany, May 2005.

[Sophos2003] Sophos.com News Story, "Laptops may bring Blaster disaster, Sophos Anti-Virus issues advice" http://www.sophos.com/pressoffice/news/articles/2003/08/va_blasterlaptop.html August 14, 2003. Retrieved on November 24, 2007.

[Stajano1999] Stajano, F., Anderson, R. "*The Resurrecting Duckling: Security Issues for Ad hoc Wireless Networks.*" 3rd AT&T Software Symposium, Middletown, NJ, October 1999.

[SWIPL] SWI-Prolog http://www.swi-prolog.org/ Retrieved September 27, 2007.

[Tanenbaum1986] Tanenbaum, A. S., Mullender, S. J., and van Renesse, R., "Using sparse capabilities in a distributed operating system." *Proceedings of the 6th International Conference on Distributed Computing Systems* (ICDCS), pages 558-563.

[TCG-NC] Trusted Network Connect Working Group. https://www.trustedcomputinggroup.org/groups/network/ Retrieved November 1, 2007.

[TCPA] Trusted Computing Platform Alliance http://www.trustedcomputing.org

[Tourrilhes1997] Tourrilhes, J. The Linux Wireless Extensions. http://www.hpl.hp.com/personal/Jean_Tourrilhes/Linux/Linux.Wireless.Extensions.html Retrieved November 9, 2007.

[Tuchinda2002] Tuchinda, R. Access Control Mechanism for Intelligent Environments. Bitstream, the MIT Journal of EECS Student Research. Spring 2002.

[UCI2003] Microsoft RPC Exploit & W32.Blaster.Worm http://www.nacs.uci.edu/security/MicrosoftRPCExploit.html Retrieved on November 24, 2007.

[Undercoffer2003] Undercoffer, J., Perich, F., Cedilnik, A., Kagal, L. and Joshi, A. "A Secure Infrastructure for Service Discovery and Access in Pervasive Computing," Article, ACM Monet: Special Issue on Security in Mobile Computing Environments, October 2003.

[UPnP] Universal Plug and Play Consortium. http://www.upnp.org

[Waldo1999] Waldo, J. "The Jini Architecture for Network-Centric Computing," *Communications of the ACM*, Vol. 42, No. 7, pages 76-82, 1999.

[Wang2004] Wang, B., Bodily, J., Gupta, S. "Supporting Persistent Social Groups in Ubiquitous Computing Environments Using Context-Aware Ephemeral Group Service". *Proceedings of the Second IEEE International Conference on Pervasive Computing and Communications* (PerCom'04).

[Weiser1991] Weiser, M. "The Computer for the 21st Century." Scientific American 265(30), pg. 94-104, 1991.

[X509-AM] ISO/IEC JTC1/SC 21, Draft Amendments DAM 4 to ISO/IEC 9594-2, DAM 2 to ISO/IEC 9594-6, DAM 1 to ISO/IEC 9594-7, and DAM 1 to ISO/IEC 9594-8 on Certificate Extensions, 1 December, 1996.

[XACML] http://java.sun.com/developer/technicalArticles/Security/xacml/xacml.html

[XDI] http://www.xdi.org OASIS XRI Data Interchange (XDI)

[Yixin2006] Yixin Jing, Dongwon Jeong, Jinhyung Kim, Doo-Kwon Baik: A Rule-Based Publish-Subscribe Message Routing System for Ubiquitous Computing. In the Proceedings of the 3rd International Symposium on Ubiquitous Computing Systems (UCS 2006).