An Extensible Set-Top Box Platform for Home Media Applications

W. Keith Edwards, Mark W. Newman, Trevor F. Smith, Jana Sedivy, Shahram Izadi

Abstract — *The number of different types of devices on the home network is expanding greatly. While this explosion of innovation provides compelling new devices to consumers, it problems with ensuring compatibility among these devices, and providing a useful overall user interface for them. This paper describes an experimental set-top box platform for home media applications. The key feature of this platform is that it can dynamically exchange code with other devices on the home network, allowing it to not only acquire new capabilities at runtime, but also provide new capabilities to peer devices on the network. These new capabilities can take the form of new data transmission protocols, CODECs, or user interfaces. We describe the underlying architecture of our platform, as well as the user interface that allows control over a variety of home devices.*[1]

Index Terms — set-top box, home network, mobile code, user interfaces.

I. INTRODUCTION

The home network is rapidly increasing in complexity: every year brings an expanding array of connected consumer electronics devices, intended for deployment on the home network. While this rapid churn delivers innovative devices to the consumer, it also brings a number of problems, particularly: (1) how to achieve interoperability among the expanding array of devices, and (2) how to achieve a unified point of control for this expanding array of devices.

In a relatively "closed" environment—one serviced by a single vendor, or one in which all devices on the network can be co-developed to be compatible and present a unified user interface—these problems are less pressing. The home, however, is a deeply heterogeneous environment, combining devices from multiple vendors, from different hardware generations, added to the home over time. In such a setting, problems arise of how to achieve compatibility while allowing evolution, and how to achieve a consistent and unified user experience.

One of the chief architectural reasons these problems exist is that, in most connected device platforms today, the software both devices involved in an interaction must be explicitly written to know about each other. For example, in the UPnP platform, a control point that interacts with a MediaServer device must be written to know about this specific type of device. New types of devices that appear on the network would not be usable by this control point until it had been updated to work with them. This dependency on knowledge of peer devices limits the degree to which compatibility can be maintained in the presence of an evolving network of device types. It also restricts the creation of holistic control user interfaces to situations in which all device types are known ahead of time.

We have been exploring solutions to these issues through an experimental middleware platform for extensible interoperability, called *Obje*. This platform is intended for a range of applications, including especially home media applications. (See [3][4] for a description of a number of other applications of the technology, outside of the home media domain.)

This paper describes a prototype set-top box (STB) application built using the Obje middleware. This STB application runs on commodity (PC-like) hardware, as well as a number of small footprint commercial systems such as home server appliances, and Pocket PCs. It provides a number of traditional media-oriented STB services, and also acts as a unified point of control for other devices and services on the home network. One of the key features of the STB is that it can dynamically adapt to the presence of new, Obje-compatible devices through a mobile code mechanism, that allows the STB to acquire new media handling behaviors necessary for interactions with new devices.

The next sections describe the underlying middleware system used by the set-top box, as well as its architecture and interface, and details of its implementation.

II. MIDDLEWARE PLATFORM

The most distinguishing feature of our middleware platform is that it allows runtime extensibility of devices and applications, allowing new devices that enter the network to provide code to peers to allow them to interoperate with the new device.

This functionality is exposed as a *bootstrap protocol* layered atop TCP/IP, which provides a number of operations designed to support runtime extensibility. Most importantly, the protocol allows a new device on the network to provide a peer device with:

[1] This work was supported in part by the U.S. National Institute of Standards and Technology (NIST).

W. Keith Edwards is currently at the Georgia Institute of Technology, Atlanta, GA 30332-0760 USA (email: keith@cc.gatech.edu). He was previously at the Palo Alto Research Center (PARC).

Mark W. Newman and Trevor Smith are at the Palo Alto Research Center (PARC), Palo Alto, CA 94304 USA (email: mnewman@parc.com, tfsmith@parc.com).

Jana Z. Sedivy was formerly at the Palo Alto Research Center; she now resides in Ottawa, Ontario, Canada (email: janasedivy@yahoo.com).

Shahram Izadi is at Microsoft Research, Cambridge, UK (email: shahrami@microsoft.com). He was formerly at the Palo Alto Research Center (PARC).

Contributed Paper
Manuscript received August 3, 2005

- An implementation of a new network protocol needed to communicate with the device.
- An implementation of one or more CODECs, to render or process media received from the new device
- An implementation of new user interface (UI) controls, which can be used to control the device remotely

We call this protocol a *bootstrap* protocol because it is used for the initial negotiation and transfer of new capabilities necessary for compatible communication. Once this transfer has been completed, two devices communicate with each other directly, using these new capabilities.

These implementations of new capabilities are in the form of mobile code—self-contained executable content delivered over the network to the peer device. Our middleware platform allows for a variety of code formats, including platform-independent code (e.g., Java [1]) as well as highly-tuned, platform-specific code (which of course would only be executable on a compatible target device).

Obje devices or services (which we generically call *components*) must carry an implementation of the bootstrap protocol, may have one or more versions of mobile code intended for user by peers (these would typically be carried in some form of stable storage, such as firmware, flash, or on a disk), and may have the ability to execute code received over the network.

I. Bootstrap Protocol and Code Formats

The Obje bootstrap protocol is defined as a profile on the Blocks Extensible Exchange Protocol (BEEP) [11], a generic application protocol kernel for bidirectional, connection-oriented messages. BEEP provides facilities for TLS-based security, providing message integrity and privacy, as well as authentication of peers on the network.

Devices advertise their presence over the local link via multicast DNS; these advertisements take the form of Uniform Resource Identifiers (URIs) that indicate the address of the device. Once a URI for a given device has been discovered, an Obje peer may communicate with it using the bootstrap protocol. A message called FetchRequest returns ComponentDescriptors for the discrete services running at that device. ComponentDescriptors are short XML documents that provide descriptive information about a component (name, icon, and so forth) as well as information about the roles a component may play (source of data, recipient of data, and so forth), and any mobile code that may be provided by the component.

We call stand-alone bundled of mobile code *granules*, and they are represented in the protocol via elements called GranuleDescriptors. Each GranuleDescriptor indicates a location from which the mobile code may be loaded (typically from the device itself), as well as parameters used to initialize loaded code granules, a unique version identifier that may be used by clients to cache code granules, and a specification of the platform requirements of the code granules.

Devices that can send or receive media content declare in their ComponentDescriptors any content types that they may be able to process "natively"—meaning, without the need to acquire any code granules from a peer. These declarations are in the standard MIME format [2], and provide a mechanism for common content types to be handled without the need for mobile code.

Depending on the role a component plays, it may provide a number of granules for specialized uses. For example, components that can act as originators of data (called DataSources) may be able to transmit specialized granules that provide new protocol implementations or new CODEC implementations. Other sorts of components may provide custom UI implementations or custom discovery protocol implementations. There are a fixed number of component roles defined by Obje, and thus a fixed number of granule types.

Once a granule, such as a new protocol implementation, is transferred to a peer, it can be executed directly by that peer. Thus, after the initial bootstrap phase to exchange any code necessary for compatibility, two Obje peers can communicate directly with one another, using the protocol and data types provided by the source component. This further communication does not involve the bootstrap protocol itself.

Effectively, this mechanism allows peers to be built against a static protocol specification (the bootstrap protocol), which is then used to exchange new capabilities necessary for compatibility, as new peers enter the network.

In our current implementation, granules are not cached. In other words, code is transferred between devices at the start of each pairwise interaction. This arrangement is well-suited for devices with low memory capacities; other systems, however, may cache granules to save on network bandwidth and transmission time at the expense of storage space.

As noted earlier, any given device may provide granules in multiple executable formats—a platform-indepenent version and a more highly tuned platform-dependent version, for example. These executable formats are named via simple strings that indicate the intended platform, such as "java-jdk1.2" or "x86-win32."

II. Benefits

By shifting the burden of agreement from development time to runtime, we can achieve a number of important benefits that are especially useful in the home media setting.

For example, a vendor may produce a new, highly-efficient data communication protocol tuned for a certain class of media. Normally, delivering such a new protocol in the marketplace would likely require standardization, buy-in from other vendors, and time for implementations of the protocol to become widespread on deployed devices. Under our dynamic extension architecture, such a vendor could create such a new protocol, include an implementation of it on a new device, and have that device deliver the protocol to any peers equipped to speak the Obje bootstrap protocol. This arrangement effectively allows the device to be able to use its own custom code at both endpoints of the communication channel.

This same mechanism can also be used for CODEC implementations. A device that provides a new media encoding can deliver the capability to decode and render this

format to peer devices, as needed. This arrangement can allow more rapid experimentation and delivery of new media formats into the home network.

In situations where a vendor may wish to exercise control over which devices may be able to accept a certain media stream, necessary granules may be encrypted, allowing only peers with the necessary key to obtain and load the code necessary to process that media.

III. An Experimental Set-Top Box

We have used the platform described in the previous section to build an experimental home media hub—a set-top box that (a) provides a number of services that are accessible throughout the home network, (b) can discover and use other devices that enter the home network, and (c) provides a unified UI for interacting with and combining the various services and devices throughout the home.

Our initial implementation of this experimental box has focused on the software aspects of extensible interoperability. Thus, our system has been developed on commodity hardware: a small footprint, PC-based system with onboard video hardware, hard disk-based storage, infrared receiver, USB and IEEE 1394 ports, and DVD playback capability. While this development hardware provides perhaps more computational power than many current commercial STBs, it serves as an excellent development environment for software prototyping. Since our initial prototype, we have successfully deployed the platform on less powerful devices, such as commercial home gateway/server appliance and various WindowsCE devices.

Run on the commodity development hardware, the system exposes a number of services to the home network:

- Storage (for audio, video, and image collections)
- DVD playback
- Television/cable tuner and digitization/encoding
- Audio playback (when connected to speakers)
- Video playback (when connected to a television or other monitor)
- Connectivity to devices through USB and IEEE 1394 ports

Many of these services are found on current commercial set-top boxes; indeed, when used as a standalone device, this prototype emulates the functionality of current STBs. For example, media stored on the box can be rendered to attached speakers or monitor.

However, because all of box's services is implemented as discrete Obje components, these services can be individually discovered, used by, and combined with other devices on the home network. For example, the audio and video outputs of the box are exposed as a network-accessible service, and can be used to render content from arbitrary Obje-enabled devices on the home network. Likewise, incoming content (such as from a ripped DVD, or digitized television program) can be streamed over the network to a storage device connected on a PC.

Perhaps more importantly, the extensibility features of our platform allow the services on the box to be used with entirely new types of devices that enter the network (as long as those devices can communicate using the Obje bootstrap protocol). For example, a new digital audio player device may use an audio encoding format unknown to the STB. Such a device would be created by its developers to carry an implementation of a CODEC needed to process the media encoding; optionally, the developers of the device may provide a number of implementations of the CODEC, such as a platform-neutral implementation (which perhaps performs slowly) and one or more tuned platform-specific implementations. These latter would likely be created to target common hardware platforms (such as Windows) or the vendor's own hardware families.

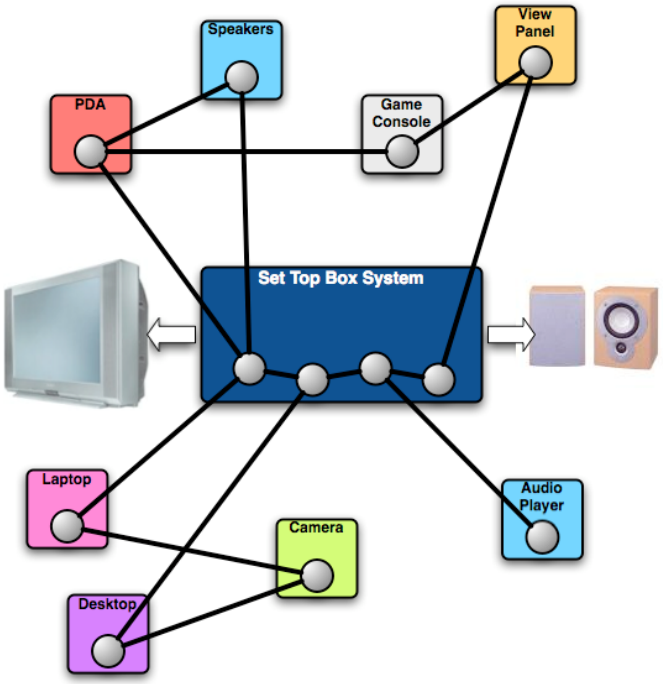

Fig. 1. The extensible STB sits at the hub of the home, exchanging capabilities as needed with new devices that enter the network. Media from remote devices can be streamed to the monitor and speakers connected to the STB; likewise, content stored on the STB can be streamed to networked devices throughout the home.

When connected to the network, the digital audio player would appear to the STB as an audio source. If a user initiates a connection to the digital audio player (as described in the section, *User Interface*, below), the DAP would first negotiate with the STB on which executable formats it can process, then transfer the necessary CODEC via the bootstrap protocol. After this point, the audio is streamed and rendered by the STB, using the newly acquired CODEC.

In a sense the capabilities of the STB are "exploded," incorporating devices elsewhere on the home network, and able to be freely recombined with those devices. New devices carry with them the behaviors needed for the STB to flexibly communicate with them and deal with their media formats, allowing these devices to be incorporated into the home network.

Figure 1 illustrates this concept. Here, the STB is at the hub of an expanding and *open-ended* array of devices. The components hosted on the STB communicate among themselves, and with off-board devices, using the same extensible mechanisms. The UI provided by the STB aggregates these devices into a consistent interface.

I. Set-Top Box User Interface

Our experimental STB platform provides an animated graphical user interface (GUI) that is displayed on the connected television, and can be controlled with an included IR remote. This GUI allows users to interact with not only the on-board services on the STB itself, but also with devices discovered on the home network. Further, the GUI is dynamically modified in response to the comings and goings of devices on the network; for example, as new sources of audio appear (such as connected digital audio players), the menus adapt appropriately.

Figure 2 shows the main screen of the GUI, for a basic configuration of devices on the network. The main menu items include:

- Watch TV
- Audio Library
- Video Library
- Image Library
- DVD
- Internet Radio

When no other devices are present on the network—meaning that the STB is operating as a standalone appliance—these menu items connect services running on the STB itself. For example, the "Watch TV" item connects the television tuner service to the video display service that drives the external monitor. Likewise, the various "Library" menu options allow connection of the STB's onboard storage service to onboard video and audio output services. These within-box connections use the same Obje mechanisms as do outside-of-box connections.

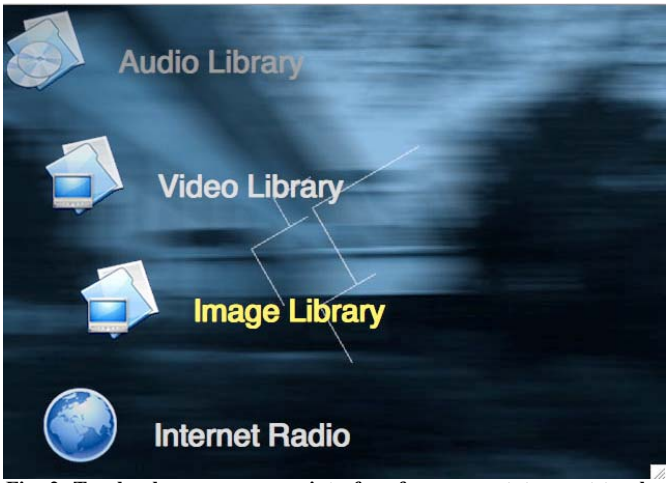

Fig. 2. Top-level on-screen user interface for our prototype set-top box. Users interact with the controls via an IR remote.

When external devices are present on the home network, however, these menu items adapt to their presence. For example, as other devices capable of displaying content become available—meaning, devices that can play the role of data recipient for visual data, and can either accept streaming video data in the formats provided by the STB or can execute granules provided by the STB to allow them to do so—selecting the Watch TV menu will bring up a list of possible recipient devices throughout the home.

Likewise, if new devices that store audio, video, or image content—such as digital audio players or cameras—are connected, then selecting one of the Library menu options will bring up a list of discovered content sources. After selecting one, the system allows the user to select a target recipient—either the onboard monitor and speaker outputs, or compatible recipients discovered on the network. Figure 3 shows the user interface after a sound file has been selected; here, it can be played through the connected speakers, to transferred to a "File Space" storage service elsewhere on the network.

This UI design embodies a number of concepts we believe are important. First, the top-level menus are relatively stable, ensuring predictable operation to allow users to easily learn the system. Second, in simple cases—where the endpoints of an interaction are unambiguous—the system implicitly assumes these endpoints rather than asking the user. Finally, the system adapts the UI in specific ways to the presence of new devices. This approach allows the user to select first a source, and then the system filters and presents compatible potential destinations in a menu.

Fig. 3. When multiple devices are present on the network, the STB presents the user with a choice for where to send media content. Here, the connected speakers and a remote "File Space" service are available. If other STBs, or other Obje-compatible speakers or devices were present, they would be presented here to.

II. Bridging to Non-Obje Devices

While our framework provides a platform for extensible interoperability of Obje-enabled devices, we realize that no devices outside our laboratory are equipped with our experimental middleware. Thus, an essential aspect of our STB architecture is a facility to bridge "legacy" (meaning non-Obje) devices into the network.

The STB software platform is equipped with an adapter module that can transparently bridge a number of common

hardware classes into the Obje environment. This adapter detects the presence of a range of USB devices connected to the STB, and creates virtual Obje devices that represent these. Currently this support is limited to USB cameras and digital audio players, though it could be extended to other sorts of USB devices, as well as non-USB devices (such as Bluetooth and IEEE 1394 peripherals).

These virtual devices appear on the network as discrete, discoverable entities, and work in the same was as native Obje devices do. The STB adapter module communicates with peers using the bootstrap protocol on behalf of these virtual devices, effectively allowing devices with extremely limited computation capabilities and no native networking capabilities to be usable on the network as full-fledged Obje peers.

For example, a digital audio player connected to the STB appears as an audio source in the STB's menus, and is also discoverable by other devices elsewhere on the network, and usable by them. Audio can be streamed from the connected player over the network to other STBs, to PCs or laptops, or to network-connected speakers.

We have also isolated this adapter module into a standalone software installation for PCs and laptops, allowing them to proxy for devices connected to their USB ports onto the Obje network. Thus, a digital camera connected to a laptop will appear on the STBs menus, as well as being accessible to any other Obje device on the network.

IV. Implementation

As noted earlier, our current hardware platform for the STB is a small form-factor PC, to allow easier development of our software infrastructure. Our software system has been ported to a number of smaller devices, however, including a commercial home gateway/server appliance and the WindowsCE environment.

The STB software is implemented in Java, with platform-specific extensions to handle interaction with I/O devices on the hardware platform. These include interactions with the television tuner, DVD player, and hardware-assisted video CODEC and rendering card. The STB software itself is approximately 1.9MB, including the services hosted on the box, but of course also requires the bootstrap protocol implementation as well as a JVM.

The current bootstrap protocol implementation is also in Java, and is approximately 1.3MB including all necessary libraries.

While we have targeted Java for portability and speed of development, nothing in either the STB platform or bootstrap protocol require Java. Alternative implementations could be created in other languages, potentially with lower storage, memory, and computation requirements.

Our platform's architecture imposes an initial startup latency when two peers begin communication; this is the period during which initial negotiation happens, and any necessary transfer of mobile code takes place. The process requires two roundtrips after discovery (once to request and receive the device's ComponentDescriptor, and once to request and receive any needed mobile code). This exchange happens only once per connection between peers, and is generally not perceptible to the end-user, as most message payloads are on the order of hundreds of bytes, and most code granules in our prototype system are on the order of a few kilobytes. Once a code granule has been transmitted and loaded, the bootstrap protocol is no longer involved, and performance is determined by the efficiency of whatever per-device transfer protocol the device developer may have created, as well as that of the underlying physical transport. In other words, after the initial negotiation and code loading phase, performance can be the *same* as a device developer would have provided natively.

V. Related Work

A number of systems, from both the research and commercial spaces, have explored platforms for networked media in the home. The Universal Plug and Play [5] platform, for example, uses a combination of "web-friendly" protocols (SOAP, HTTP, RTP) coupled with the standardization of device type-specific profiles (for MediaServers, MediaRenderers, Scanners, Printers, and so forth) to support networked home media services. One crucial difference between our platform and UPnP is that UPnP requires agreement on device profiles, protocols, and media types be built into all communicating devices at development time; our platform requires only base-level agreements (for the bootstrap protocol) and defers other mechanisms necessary for communication until runtime.

The Jini platform [12] is perhaps closest to ours in spirit, as it relies on mobile code for dynamic exchange of object implementations at runtime. Jini itself, however, does not combine this mechanism with an architecture specifically designed to support media exchange; it does not, for example, define common interfaces for new protocol handlers or CODECs. Also, Jini—as a Java-centric platform—does not allow the easy delivery of non-Java mobile code, nor does it lend itself well to non-Java implementations, for footprint or performance reasons.

Systems built around the Open Services Gateway Initiative (OSGi) [10], such as [7][9] are analogous to ours in a number of ways. First, they rely on a connected gateway box as a centralized hub for home management and control. Second, they have the ability to download new "bundles" of executable content from a remote service provider, allowing them to extend their functionality to new circumstances. One chief difference between OSGi-based systems and ours, however, is the fact that our platform supports peer-to-peer delivery of new capabilities, allowing new devices that enter the network to immediately extend their peers to be able to interact with them.

From the research community, Stanford's iRoom system [6] also supports dynamic interactions among devices and services. Their system, however, is not intended specifically for home media applications. Their architecture is focused primarily on exchanging of control and event data through a "tuple space" mechanism, rather than extensible streaming media data.

HP's Cooltown system [8] likewise provides the ability for ad hoc interactions among devices and services, using a web-centric framework of protocols. While Cooltown leverages the advantages of the web (simplicity, familiarity to developers), it is also bound by the limitations of the web (limited range of acceptable protocols and media formats, in particular), which may weaken its appeal for streaming media applications.

VI. CONCLUSIONS

This paper has described a prototype software platform for extensible set-top boxes, which aims to overcome some of the limitations of current systems. Namely, our platform allows devices on the home network to extend each others capabilities, preserving compatibility as new sorts of devices enter the network. We believe that this approach has the potential to reduce consumers' frustrations with incompatibilities, repeated software updates and driver downloads, by allowing decentralized distribution of necessary software components at runtime.

This approach also has the potential to allow much more lightweight experimentation with—and introduction of—new media formats into the home network. By removing the requirement that peer devices agree on specific content types (which usually requires a standardization process), vendors can more easily deploy custom content encodings, with their required protocols and CODECs, and yet retain interoperability; rather than requiring agreement on the content encodings themselves, our approach requires only agreement on the initial bootstrap protocol. Approaches such as this have the potential to allow greater differentiation and innovation without incurring the loss of compatibility, which is usually a cost of evolution on the network.

The architecture of the system does impose a number of costs, however. The chief among these is the requirement that devices be able to accept and execute mobile code from peers in order to achieve the full benefits of extensibility. Obviously, devices that can provide and accept platform-independent code deliver the broadest extensibility, but this requires that they include an execution environment for Java bytecodes or other similar portable execution formats. The system does fully support native granules, which can potentially be executed by peers without the overhead of a portable execution environment, and potentially at higher performance; in such a case, however, extensibility is limited to those devices that can exchange compatible code granules.

The system's reliance on mobile code also raises security issues, of course. There are a number of solutions to this problem, the most direct of which is for devices to only accept granules signed by known providers. Thus, a vendor could create a compatible, yet extensible, family of products that would only accept "known safe" granules from products signed by that vendor. Other approaches are possible, including preventing mobile code loading from devices that are not part of a restricted trust group, perhaps defined by the owner of the devices—for example, to create a trusted group of all of the devices within one household.

As noted earlier, another potential risk is through unwanted distribution of CODECs or other code to unlicensed devices on the network. Similar mechanisms can be used to protect against such distribution, by encrypting code granules so that they can only be decrypted (and thus, executed) on licensed devices that possess the necessary key.

We are continuing our work to refine other aspects of our STB platform and core protocol as well. One possibility that we believe holds great promise is to use the granule capability to deliver new Digital Rights Management (DRM) implementations to peer devices. In much the same way that Obje devices can deliver custom CODECs necessary to render the media content they provide, these devices could also provide custom DRM implementations, securing delivered content, and providing whatever access rights are appropriate to that content. In our model, the device that delivers the content has opportunity to deliver the code that handles this content, effectively allowing the content provider to control both endpoints of the communications channel.

REFERENCES

[1] K. Arnold, J. Gosling, D. Holmes. 2000. The Java Programming Language, Third Edition. Addison-Wesley, Sun Microsystems Press. June 2000.

[2] N. Borenstein. and N. Freed. 1992. MIME (Multipurpose Internet Mail Extensions): Mechanisms for Specifying and Describing the Format of Internet Messages. Internet RFC 1341.

[3] W. K. Edwards, M. W. Newman, J. Z. Sedivy, T. F. Smith, and S. Izadi. 2002. Challenge: Recombinant Computing and the Speakeasy Approach. In Proceedings of the The Eighth ACM International Conference on Mobile Computing and Networking (Mobicom 2002), Atlanta, GA USA, September 23-28, 2002.

[4] W. K. Edwards, Mark W. Newman, Jana Z. Sedivy, Trevor F. Smith. 2004. Supporting Serendipitous Integration in Mobile Computing Environments. International Journal of Human-Computer Studies, Vol. 60, No. 5-6, pp. 666-700. May, 2004.

[5] M. Jeronimo and J. Weast. 2003. UPnP Design by Example. Intel Press, 2003.

[6] B. Johanson, A. Fox, and T. Winograd. 2002. The Interactive Workspaces Project: Experiences with Ubiquitous Computing Rooms. IEEE Pervasive Computing Vol. 1, No. 2, pp71-78. 2002.

[7] D. O. Kang, K. Kang, S. Choi, J. Lee. 2005. UPnP AV Architectural Multimedia System with a Home Gateway Powered by the OSGi Platform. IEEE Transactions on Consumer Electronics, Vol. 51, No. 1, pp. 87-93, February 2005.

[8] T. Kindberg and J. Barton. 2001. A Web-Based Nomadic Computing System. Computer Networks 35, 4, 443-456.

[9] X. Li and W. Zhang. 2004. The Design and Implementation of Home Network System Using OSGi Compliant Middeware. IEEE Transactions on Consumer Electronics, Vol. 50, No. 2, pp. 528-534. May 2004.

[10] OSGI Alliance. 2003. OSGi Service Platform, Release 3. IOS Press, Amsterdam, The Netherlands. 2003.

[11] M. Rose. 2001. RFC 3080: The Blocks Extensible Exchange Protocol Core. Internet Engineering Task Force (IETF). March, 2001.

[12] J. Waldo. 1999. The Jini Architecture for Network-centric Computing. Communications of the ACM. pp. 76-82. July, 1999.

Keith Edwards is an Associate Professor at the Georgia Tech College of Computing, where his research applies an HCI focus to problems of networking and security. Prior to joining Georgia Tech he was a Principal Scientist at the Palo Alto Research Center (PARC) where he started and led the Speakeasy/Obje project, and was involved in a number of other efforts combining human-computer interaction research with systems and networking infrastructure research.

Mark W. Newman is a research scientist at the Palo Alto Research Center and a doctoral student in computer science at the University of California Berkeley. Since joining PARC in 2000, he has worked as part of the Obje project to explore how users interact with ubiquitous computing environments, and how systems can be designed to better support those interactions. Mark was a founding member of UC Berkeley's Group for User Interface Research, where he was a key contributor to the design and development of DENIM and the Designer's Outpost--two systems that helped define the space of "informal user interfaces."

Jana Z. Sedivy .was a research scientist at PARC from between 2000 and 2005, during which time she was a key contributor to the development of the Obje Interoperability Framework. Her research spans the intersection of Ubiquitous Computing and Human Computer Interaction. She currently resides in Ottowa, Ontario and serves on the board of directors for the World Computer Exchange.

Trevor F. Smith is a member of the ubicomp group in the computing science laboratory at the Palo Alto Research Center.

Shahram Izadi is a research scientist within the Computer Mediated Living Group at Microsoft Research, Cambridge UK. His research centers on enabling novel user experiences in a world of richly interconnected devices and software services. His PhD, awarded at the University of Nottingham, explored interaction with diverse display technologies, in particular the intersection between large situated displays and mobile devices. Shahram has been involved in a number of research projects including Equator and Dynamo, through which he has created a variety of applications, and UI and middleware toolkits.