Smart Office Spaces

Bora A. Akyol, Matt Fredette, Alden W. Jackson, Rajesh Krishnan,
David Mankins, Craig Partridge, Nicholas Shectman and Gregory D. Troxel.
BBN Technologies
smart-spaces@ir.bbn.com

Abstract

Imagine a world in which every device has an embedded processor and a high-speed wireless link. Any two devices can talk to each other and you link devices together as needed to get your work done.

Devices with embedded processors and wireless links are coming soon. This paper looks at some of the problems we have to overcome to make it possible to link devices together and get work done (rather than cause frustration).

1 Introduction

The advent of embedded processors with built-in wireless transceivers offers a number of chances to revolutionize both communications and computation. We use the term *smart space* to refer to the environment created by a cluster of these wireless-connected processors. At BBN we are investigating the problems of using smart spaces to revolutionize the office environment.

We chose to focus on the office environment for three reasons. First, the overall goal is clear: to enable an office worker to work more effectively with whatever electronic tools are in proximity to him or her. Smart spaces must make work easier (or more straightforward) than current systems or they will not succeed in the office environment.

The second reason for our focus on the office is that it clarifies the role of the Internet. The Internet is one office tool among many; a very important tool, but not the only one. Leslie Lamport once said, "A distributed system is one in which the failure of a computer you didn't even know existed can render your own computer unusable," [Lamport]. Businesses spend a lot of money to try to avoid this kind of scenario. At the same time, the profusion of devices in a smart space means the likelihood that some device will fail is high.

The Internet will not always be accessible to a smart space. For example, even users of today's low speed cellular phone networks periodically suffer poor signal quality. So our goal is to create smart spaces that can capitalize on the Internet's resources, when available, but still allow users to work productively when the Internet is not available. A corollary is that Internet access should not be required to complete a purely local transaction.

The third reason to focus on the office is that, being office workers ourselves, we can try to live in the smart spaces we build. Living a vision is a very effective way to find mistakes and identify unexpected benefits.

In the rest of this paper we present what we believe are the key challenges to incorporating smart spaces into the office and then discuss potential solutions that we are exploring to two of those challenges.

2 What is a Smart Space?

Before examining the challenges to creating a smart space in an office, discussion of what constitutes a smart space is worthwhile.

In our vision, every device such as a display, a mouse, a keyboard or a disk drive, contains its own embedded processor and is wireless capable. In such a world, assembling a computing environment should be a matter of placing the necessary parts within wireless signaling distance of each other. Integration of the devices into a computing environment should be transparent to the user.

In an office environment, there is also a premium on minimizing the work necessary to achieve a result. For instance, we believe that assembling a computing environment in an office is a matter of collecting the devices necessary to perform the current task and we should seek to minimize the number of devices necessary to achieve any one task. For instance, in our view, displaying a file (to be read, or projected in front of an audience) should require the presence of just a display device, a storage device containing the file, and some interface (perhaps a button already on the display device) to enable scrolling forward and backwards in the document. In a wilder example, a confident touch typist should be able to edit a file if she has access to a keyboard and the disk the file is on.

3 Challenges

We see four broad challenges to creating effective smart spaces in an office environment. All four challenges involve eliminating dependencies that hinder flexibility. In a smart space, with a potentially large number of components, even a modest failure rate can lead to a system where the inability to do work is chronic. One way to avoid this problem is to reduce failure rates (which is often hard). Another approach, which we are pursuing, is to vigorously avoid creating dependencies.

The four challenges (all forms of dependencies) that concern us are:

Power dependencies: Power is obviously a pressing concern in smart spaces. With dozens of devices cooperating, the likelihood (with today's technology) that one of them will have a battery running low or will need to be plugged in, is high. Another way to think of this problem is that fully untethered dynamic computing requires us to get rid of two wires: the communications wire and the power wire. Embedded wireless gets rid of the communications wire. We still need to worry about power.

Network dependencies: A lot of problems in smart spaces get easier if you assume that a smart device is constantly attached to the wider Internet. In particular, it is easy to assume that smart devices are configured by means of a network device configuration protocol such as the Dynamic Host Configuration Protocol (DHCP). Furthermore, if a device lacks some piece of information (an applet, some data, or a name-to-address binding) it is often convenient to assume that this information can be retrieved from some repository. Authentication of a user or a device may require an authentication certificate to be downloaded from a certificate authority.

But there are a number of environments where devices may not be connected to the Internet, or may have only intermittent connectivity. In our view, it is unacceptable to design a system that requires Internet connectivity to function correctly.

Peripheral dependencies: It is very easy to build in dependencies on certain types of peripherals. The most obvious example is the keyboard. Many applications unnecessarily assume the user has a keyboard. Any input device (mouse, voice input, or keyboard) should suffice, especially if the user wants to do something simple like deleting a slide from a presentation.

Application dependencies: The major reason many people haul PCs around is that the PC contains the code for all the applications they use. If we are really going to make smart spaces work, we need to solve the problem of the tight coupling between applications and the files they produce or maintain.

An example may help to illustrate this point. Suppose you are going to give a presentation. In the ultimate smart space, all that should be required is a self-reading diskette cartridge that contains a copy of the presentation, a projection device such as a flat screen or projector and some mechanism, such as a button, to advance the slides. But in today's world, this configuration isn't sufficient. You'll also need the software that generated the presentation and knows how to read and display the presentation file. We think that's undesirable: it creates a situation in which it appears to the user (who can see the diskette, projector and button) that she has everything necessary to give the presentation, but in fact it is not possible.

4 Solutions

Enabling a smart office space that empowers the people in that space to work more efficiently requires that we innovate to eliminate the dependencies on power and network connections, and the dependencies on particular peripherals and applications. In this section, we present work at BBN on solutions to power and network connection issues, and sketch some thoughts about how (or how not) to address peripheral and application dependencies.

4.1 Eliminating Power Dependencies

Our vision for a smart office space is truly tether-less. A truly tether-less environment is free from wires not only for communication but also for power.

The idea of wireless transmission of power has been around since Tesla [Brown, McSpadden]. Modern applications that have been proposed range from beaming power by microwave to aircraft to beaming solar energy from stations in outer space to the ground. Indeed one could argue that almost all wireless communications are specific cases of wireless power transmission. Several applications of this technology ranging from electric power transmission to microwave beam powered aircraft have been publicly demonstrated. However the applications considered so far have been on a grand scale at power levels of 500 watts or higher. Proposals also have been made to build generating stations, using photovoltaics and other technologies, on the moon, on nearby asteroids or satellites, from where the power would be beamed down to earth.

At the other end of the spectrum, very low power and extremely short-range wireless powered devices are already in use in medicine. In certain surgical implants it is considered risky or undesirable to include batteries – an external battery carried by the patient is used instead to power the implant wirelessly. Contactless transfer of power to isolated integrated circuits by inductive means also has been proposed [Selvidge].

With increasing power efficiencies of semiconductor computing and peripheral devices, there is an incentive to power devices directly using radio frequencies (RF) which are currently being used for communications. Examples could be electronic product labels and price tags in supermarkets where battery-free operation would be ideal or an embedded temperature sensor in a smart office. Battery-free operation is very attractive considering that there would be fewer environmental side effects.

However, as appealing as the idea of eliminating the battery sounds at first, several applications require disconnected operation both from a wired or wireless power source as well as from the data network. We also have to be careful not to replace a dependence on batteries with a dependence on irradiation. It is more appealing to seek a hybrid approach that both provides power and also recharges an internal battery through wireless means. For example, consider that solar powered watches almost always have an internal rechargeable battery.

4.1.1 Different Approaches

Wireless recharging devices do not currently exist. We are exploring two ways to create them:

***The "Diffused Glow Interior Lighting"*:** Buildings will continue to have a wired infrastructure both for power and communications. Addition of wireless power transmitters can be done at low cost.

In this approach, buildings are equipped with power transmitters to power or recharge smart devices. The transmitters omni-directionally transmit diffuse microwave radiation. Devices to be powered or recharged by wireless means have an attached rectenna to intercept the radiated power and convert it to DC.

***The "Drying Lamp" or "Microwave Oven"* :** Bulk recharging of wireless devices is an interesting application worth considering. Devices suitably equipped with rectennas can be recharged en masse by being dumped on to a table and irradiated with microwaves from a projector (drying lamp). Or the irradiation apparatus can be shielded during operation (the microwave oven), permitting higher power densities and quicker recharging. The devices to be recharged this way either need to have their electronics shielded or they can be switched off during recharging.

The implications of wireless powering and recharging are profound. We believe that this technology has a wide range of applications and can revolutionize the way in which we think about and use wireless devices. Some questions remain to be answered regarding the safety and commercial feasibility of wireless power delivery. At the same time, the basic pieces of technology required are already available today. We are working presently to put that technology together into effective demonstrations.

4.2 Eliminating Network Dependencies

A smart device must be able to communicate with other smart devices within its vicinity. This communication should not depend on whether connectivity to the broader Internet is available. Exactly how well a device will function depends, in large part, on the device's purpose. Local operations, such as file transfer or remote login, between devices in a space should work. Remote operations, such as emailing the home office or resynchronizing with an office calendar program, may not work, or may require that a request be logged, to be fulfilled when Internet connectivity is available again.

Smart devices that communicate with each other without the need for external configuration are referred to as an ad-hoc network (AN). In order to enable ad-hoc networking of smart devices, the wireless channel that these devices share must support a discovery mode and must allow for arbitration of resources. Once the smart devices discover each other, they should be able to communicate without external support. At the same time, if one of the devices has Internet connectivity, Internet access should be available to the entire AN, without requiring the AN to reconfigure itself.

A major challenge is self-organization – enabling the smart devices to form a useful ad-hoc network quickly. It should take very little human input and allow the configuration effort to scale as the number of devices or spaces or both grow large. The mechanisms to create the ad-hoc environment should also be parsimonious in their use of bandwidth. We want to use most of the network's bandwidth to do work, not manage the network.

4.2.1 Security and POKI

Security usually requires the labor-intensive process of integrating a new device into a public-key infrastructure (PKI), so it can use public-key authentication.

Current and proposed public-key infrastructure such as X.509v3 [Housley] and SPKI [Lampson] focus on achieving high assurance of name-key bindings while accepting moderate administrative costs for activities such as key transport to certificate authorities, and the effort to verify identities. While there will always be scenarios which require these high-assurance PKIs, we believe Smart Spaces can not rely on them because the cost of manually configuring every smart device outweighs the need for high assurance. Furthermore, we believe that productive work among local smart space devices must be possible without a access to certificate hierarchies outside the space. For example, using a PKI that requires Internet or Intranet access to verify a name-key binding is not consistent with our design philosophy.

We believe that a new PKI can be designed that

1. needs no pre-configuration,
2. exchanges key information when learning about device capabilities,
3. allows varying degrees of assurance about keys bound to a name, and
4. allows automatic update of assurance estimates about keys.

First, let us look at an example of why increasing assurance will work for Smart Spaces, by investigating what happens when the residents of a neighborhood meet a new neighbor. At first the new neighbor is outwardly accepted at face value, but a resident does not really have any assurance that the new neighbor is trustworthy or even owns the house. Over time, as the putative new neighbor continues to act like the new neighbor, the estimate of assurance goes up.

In neighborhoods, people also talk about each other, and especially about a new neighbor. A new neighbor is likely to be described (and in the digital world, this would include a public key) to other residents who have yet to meet him. Thus, by the time the n^{th} old resident meets the new neighbor, their estimate of assurance of the new person's identity (the public key belonging to the new resident is K) is already fairly high.

While some PKI systems such as PGP [Callas] and X.509 with cross-certification do this with mechanisms likely to involve humans, we describe a mechanism that functions without human intervention, and so makes good sense for Smart Spaces. Our Probabilistic, Opportunistic Key Infrastructure (POKI), defines how nodes go about creating a security infrastructure simply by communicating with one another.

When two devices that speak POKI meet, they both exchange keys, and then gossip. The gossip they exchange is the names of other devices that they know, the keys that they associate with those names, and their estimate of assurance that each key is correct. In addition, they may also choose to gossip about what *other* nodes' estimates of assurance are in the different keys.

After the two nodes have exchanged all of the information (and performed the transaction the connection was started for), they execute their algorithms for updating their beliefs about names, keys, and probabilities.

With POKI, a large number of small devices can gain a moderate level of assurance about the keys in the Smart Space without pre-configuration. The gossip can occur when devices in the space inquire about each other's capabilities. Assurances that started out a low levels can increase over time without any human intervention.

4.2.2 Directories

Directories are also a good example of the need to eliminate network dependencies. Today, directories are a vital piece of network and systems infrastructures. We rely on a networked directory to translate domain names to network addresses. We rely on web client and operating system directories to tell us what application is best suited to opening a file. Yet directories are often problematic in an ad-hoc space.

The problem is best explained by examples. Suppose Cheryl and David walk into a conference room. Cheryl has a copy of a document that David needs and both PCs are part of the local ad-hoc network, but are not currently connected to the Internet. Cheryl should be able to tell David the name of her PC and he should be able to FTP a document from it. That means that the traditional domain name system lookup, which depends on the Internet's distributed domain name system, will have to be bypassed and use some local directory service to map the name of Cheryl's machine into a local address.

Now consider the problem of displaying a document from a web browsing device. A web browser typically contains an internal directory mapping file names to applications that can display them. In an ad-hoc space, the applications will generally not reside on the browsing device, but will be distributed on various storage devices, which may or may not be carried by the user of the browser. How does the browser find the right application (of the right version, that runs on the browser's processor) in the local ad hoc network?

Note that a straightforward approach, namely registering with a registry when one joins a smart space, doesn't work well. First, the assumption of a registry device introduces a dependency on the registry, both that it exists and is functioning in the space. We are just trading one dependency for another. Second, the volume of information that needs to be registered could be quite high (e.g., a list of all executable applications on a hard disk) and changing rapidly (that hard disk may have been in the pocket of someone who just walked past your conference room). Third, the bandwidth available to some of the smart devices may be low, hence making access to a registry that covers a large number of items expensive.

We suspect that the registry may have to be fully distributed. That is, all interested devices keep track of the part of the registry they care about, and devices advertise their properties to everyone. Some balance of pushing data, and querying for data when the data is needed, will have to be found. Alternatively, a service resolution protocol might be useful. A service resolution protocol multicasts requests for particular services, as needed. The advantage of such a protocol is that it is demand driven. No one needs to keep track of temperature sensors if no application is using one.

4.3 Eliminating Peripheral Dependencies

Dependencies on peripherals are often created by the programmers of applications that do not create alternate means of inputing commands or data to change the way an application behaves. For instance, in our view, if a user has access to a display and an input device, the user should be able to edit a document on the user's disk. But in today's world it is all too likely that the user's document editor requires both a mouse and a keyboard (even though one or the other is sufficient).

In order to enable smart devices to overcome peripheral dependencies, a capability description language may be appropriate. Such a language would allow the smart devices to query their peers to resolve their capabilities. So, an application might query an input device about whether it could emulate a mouse. And the slide projector in a conference room could advertise its resolution and supported scan rates.

But there's a challenge hidden in the idea of a capability language: how many different types of devices are there, and how do we represent them? For instance, a mouse can come with one to three buttons. An application will have to behave differently, depending on the number of buttons. But the last thing we want is a world where an application refuses to work because the user has a one button mouse and the application expects a three button mouse.

This problem was recognized many years ago by the ARPANET pioneers and dubbed the *m-by-n* problem. The basic idea is that if there are m applications that know the details of n different devices, then cost of adding the $n+1$st device to the mix is often prohibitive, because it requires updating the m applications. The trick is to actually have as few distinct types of devices as possible. So, for example, we could classify keyboards, pointers and mice, as input devices, capable of giving us a single one-button signal, and require every application to work this device. We can then allow optional negotiation of additional capabilities such as "can you send character codes?" and "do you have more than one button?"[1].

Developing these kinds of standards requires tremendous community effort and a great willingness to radically simplify. But it has tremendous advantages.

4.4 Eliminating Application Dependencies

As we noted above, many files are closely related to particular applications. You often cannot effectively open or manipulate a file except with the application that created it.

There are obvious solutions to eliminating application dependencies, all of which are problematic. One solution is to combine the file with its application. So every PowerPoint[2] document also includes a (platform independent) version of PowerPoint viewer. This result consumes disk space with many useless replications of our application. A slightly better approach is to put one copy of the PowerPoint viewer on any device that contains a PowerPoint file, but that still is intensive.

The UNIX solution of fairly generic text files softens the problem but does not solve it. We could, for instance, edit the source files for this paper with a range of applications (*emacs*, *vi*, *sed* and *awk* are obvious examples). But trying to format the source using *groff* instead of LaTeX would be problematic on a deadline.

New ideas are needed in order to make advances in this area.

5 Conclusion

Smart Spaces is a very rich research topic, filled with interesting problems. Our particular focus on the office environment has presented us with an inviting set of challenges to solve. We believe we have two exciting solutions: a method for wireless power delivery and a mechanism for key management in ad-hoc environments. Although we continue to be challenged by peripheral and application dependencies, we have identified problems in the obvious solutions and look forward to working on these problems more in the future.

[1] This solution is precisely the one the ARPANET designers choose for supporting terminal types over telnet

[2] PowerPoint is a trademark of the Microsoft Corporation.

References

[Brown] W. C. Brown, "The history of wireless power transmission," *Solar Energy,* Vol. 56, No. 1, pp. 3-21, 1996.

[Callas] J. Callas, L. Donnerhacke, H. Finney, and R. Thayer, "OpenPGP message format," Request for Comments (Proposed Standard) 2440, Internet Engineering Task Force, November 1998.

[Housley] R. Housley, W. Ford, T. Polk, and D. Solo, "Internet X.509 public key infrastructure certificate and CRL profile," Internet Draft, Internet Engineering Task Force, September, 1998, Work in progress, `http://www.ietf.org/internet-drafts/draft-ietf-pkix-ipki-part1-11.txt`

[Lamport] L. Lamport, Private communication to src-t bulletin board at the Systems Research Center of Digital Equipment Corporation on May 28, 1987.

[Lampson] B. Lampson, T. Ylonen, R. Rivest, W. Frantz, C. Ellison, and B. Thomas, "Simple public key certificate," Internet Draft, Internet Engineering Task Force, March 1998, Work in progress, `http://www.ietf.org/internet-drafts/draft-ietf-spki-cert-structure-05.txt`

[McSpadden] James McSpadden, "Collection of Power from Space References," `http://www.tsgc.utexas.edu/tsgc/power/general/refs.html`

[Selvidge] Charles Selvidge, Adam Malamy, and Lance Glasser, "Power and communication techniques for physically isolated integrated circuits," *Proceedings of the Stanford Conference on Advanced Research in VLSI*, pp. 231–247, May 1987.