Pervasive Computing in Health Care: Smart Spaces and Enterprise Information Systems

J. P. Black, W. Segmuller, N. Cohen, B. Leiba, A. Misra, M. R. Ebling, and E. Stern

IBM T. J. Watson Research Center

Hawthorne, NY 10532

{jpblack,ncohen,archan,ebling,estern}@us.ibm.com, {whs,leiba}@watson.ibm.com

Abstract— Middleware for pervasive computing is an active area of research that is now starting to mature. Prototypical systems have been developed and used for demonstration applications in a number of domains. However, realistic pervasive environments of the future will need to integrate with enterprise systems, including software, hardware, databases, standards, and life cycle. Designing and implementing "enterprise-strength" pervasive applications today is not an easy task, as we illustrate with a simple case study of a healthcare scenario. We use the case study to draw conclusions about requirements placed on future pervasive systems.

I. INTRODUCTION

As pervasive computing develops, there will be an evolution from isolated "smart spaces" to more integrated enterprise environments where the dream of unconstrained, ubiquitous, pervasive computing will face the realities of enterprise requirements, market forces, standardization, government regulation, security, and privacy. These tensions are particularly strong in the case of health care, given the promise of smart spaces to improve care and lower costs, the many parties with legitimate interests, and the medical and legal risks.

Our objective in this paper is to use a case study in health care to shed light on the architecture of an "enterprise-strength" pervasive-computing application, the current state of the art, and the issues that remain to be addressed before such a system becomes a practical reality. Our case study was motivated by our own research in context technologies [1]–[3], by which we mean reusable and scalable middleware to ease the design, development, and deployment of applications that sense and react to various types of contextual information as it appears and changes dynamically. As well, we wished to gain more experience with building and manipulating sources of context data.

The next section presents a vision of a "smart hospital" as a starting point for our case study. In Sect. III, we point out a number of issues that need to be addressed to achieve the vision. Sect. IV describes the various components that might serve as a high-level design for a realistic future system. Since our resources were limited, we were unable to prototype all parts of the design, and so Sect. V describes our actual prototype in more detail. Our experience with the demonstration taught us some lessons and suggested issues that remain to be addressed; these are discussed in Sect. VI. Finally, Sect. VII presents our conclusions and some outstanding issues.

II. PERVASIVE HEALTH CARE TOMORROW

We present a scenario where context middleware, within a few years, might assist in clinical-care business practices. Patients are instrumented with vital-sign monitors and with a means of determining their location. Physicians and nurses have wireless PDAs, also instrumented with a means of determining their location. Context-aware applications help optimize physician rounds, support nurse triage, simplify the user interface to pervasive devices, provide additional data for billing reconciliation, and provide clinical communications.

A physician, Dr. Able, arrives on the second floor of the hospital. A graphic appears on his PDA showing the rooms assigned to his patients. Rooms are highlighted depending on whether the patient is currently in the room, and ambulating patients are notified that the doctor is making rounds. Relatives waiting elsewhere in the hospital are also notified so that they can go to the patient's room to talk to the doctor.

As Dr. Able enters John Doe's room, a subset of John's electronic medical record appears on the doctor's screen. Current and most recent "readings" appear, with the option of confirming them as still valid, or updating them. Dr. Able's proximity to John is logged, for input to (or reconciliation with) the hospital's patient-accounting application. As the doctor enters data and records orders, logs are created for the accounting application. John's wife Joann had wanted to speak briefly to Dr. Able, and entered the request on the family patient portal, since Dr. Able had enabled that facility for her. Dr. Able's proximity to John and his location in the hospital room trigger a notification to Joann. She connects to the room and is able to take part in the visit.

Jane Smith down the hall, also one of Dr. Able's patients, experiences difficulty breathing. The patient sensors, given her medical history, trigger an alert that this is a serious condition; notifications are sent to the assigned nurse (Nurse Baker), to whatever nurse is nearest to Jane, and, since he is in the hospital, to the primary physician, Dr. Able. Nurse Baker and Dr. Able are notified when the nearest nurse enters Jane's room. If no one had, the nurse supervisor and supervising physician would have been notified.

Dr. Able continues with John. Nurse Baker views her triage list on her PDA, where her assigned patients are ordered according to an assessment of their current need for her care (from sensors, call buttons, and medical history, for example).

The ordering is determined by a combination of medical practice and hospital policy. Nurse Baker does not see an electronic chart, but rather a red-yellow-green summary of vital signs along with the triage list. Jane is clearly most in need of her help; the others are not in distress. Nurse Baker goes to check on Jane, finding on her PDA where the nearest available emergency cart is currently located, lest it be needed. Nurse Baker thanks the responding nurse, and they agree that Nurse Baker needs no further assistance from him at this time.

Dr. Able leaves John's room, and now that he is no longer "busy," all pending notifications are given. There are two, one from an outpatient with a chronic illness, who has been granted direct access to reach the doctor. The other is an important (but not urgent) lab result for Jane. He taps the lab result item, and it appears on the PDA, along with a subset of her electronic chart needed to make sense of the result. Dr. Able walks to the nursing station and to an available plasma screen there (such as a BlueBoard [4]), and touches "Start." Based on his proximity, the board presents Jane's lab result and most recent X-ray, along with a pick-list of patient data for the last three patients seen and the closest three patients. Dr. Able reviews Jane's data, and decides what to do. At this point, he tells the system to notify him immediately if any of Jane's lab results over the next twelve hours are not normal.

Context technologies are intended to support scenarios like this. "Adapters," connected to hardware sensors and software probes, gather raw context information like vital signs, the location of individuals, or their online/offline status. Higher-level "composers," developed in conjunction with medical professionals, process information from the adapters to determine conditions constituting medical alerts, summarize patient information for a nurse, or determine that a physician should be notified. Context also feeds into back-end systems like patient accounting, and causes notifications to be sent to people based on a rich representation of communication semantics. Medical device manufacturers provide context information in standardized form, and composer specifications help systematize the exchange of higher-level information. New applications are built and deployed quickly, improving patient care and reducing nurse and physician workloads for routine situations. Even so, healthcare providers remain in control and able to understand and interact with the context-aware applications and middleware.

III. PRACTICAL ISSUES IN REALISTIC PERVASIVE-COMPUTING SCENARIOS

Turning this vision into reality presents a number of significant challenges with today's communication media, hardware, and software.

Dr. Able and Nurse Baker use their PDAs for communication and as client devices for the various healthcare applications. Is this communication over a public medium, such as the (cellular) telephone network, or within some wireless LAN managed by the hospital? Can the PDAs be used outside the hospital? Which media should the PDA support? What

happens if a PDA user loses network connectivity momentarily? How do patient-monitoring devices communicate with the hospital, and how are details like the patient-device association created, updated, and dropped? Would devices for outpatients have to use different media than those for inpatients?

The scenario makes no mention of privacy, security, identification, or authentication, and yet these must be planned for and guaranteed in fairly strong ways even without considering mandated standards such as HIPAA (the Health Insurance Portability and Accountability Act) in the US.

Dr. Able's presence in John's room is noted by the smart space and linked to a back-end system for patient accounting. This implies that the smart space has some robust, reliable interface to the enterprise systems. How might these legacy back-end systems and business processes change because of the smart spaces? Might they need to be modified substantially, or is it safe to assume they all have secure but convenient interfaces accessible to the pervasive applications? How should they be architected to best incorporate and leverage the possibilities of the smart spaces? What issues of scale might the smart spaces impose on the enterprise applications?

How do Dr. Able and Nurse Baker view their day-to-day work environment? Do the BlueBoard, PDA, and conventional workstation all have the same set of applications available? Are the application user interfaces identical across devices, or even similar? Do the medical professionals have to switch devices and interfaces often? Re-authenticate often?

And finally, if an entire hospital were under the umbrella of a pervasive-computing system, with tens of thousands of devices and thousands of patients and staff, what issues of scale (beyond those of the enterprise systems) would arise?

IV. ARCHITECTURE

As we considered the overall architecture (see Fig. 1) required for a system to implement the type of scenario described above, we made some assumptions. First, we assumed that a modern, web-based, multi-tier application and development environment would continue to provide a viable base for the future system. These environments provide end-user access only through web browsers (the first tier), whose HTTP interactions are with a web and application server (the second tier), connected further with back-end enterprise applications and databases (in one or more tiers). To provide for identification, authentication, and customization by the user, we further assumed the web server was in fact a portal server, as is increasingly common in many enterprises.

End users authenticate and connect to the portal with a conventional browser, and can customize and arrange the "portlets" to which they are granted access. A portlet is a small application component displayed as part of a single browser page. One or more portlets cooperate to provide a logical pervasive-computing application. In general, the portlets display information that changes dynamically, or in response to explicit user-interface events from the user. Dynamic changes require some active component in the browser, typically involving a Java applet, JavaScript, or a similar technology.
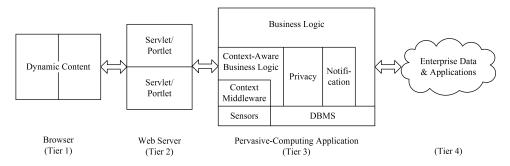
Fig. 1. Architecture of Enterprise Pervasive-Computing Applications

PDAs have different form factors, display sizes, and user interfaces than larger computers. However, we assume they run full browsers that can initiate secure portal sessions, provide JavaScript access to the browser's document object model, and execute Java applets. Browser technology and PDA capacities are improving steadily, so we feel the assumption is not overly constraining. In addition to this assumption on the browser execution environment, we assume portlet developers will leverage tools for multi-device authoring to enable access from a range of devices.

User-interface events in the portal are handled by servlets running in the execution context of the web server (the second tier). For the purposes of Fig. 1, we imagine only presentation logic in the servlets, while business logic is the responsibility of the third-tier application.

The third tier is the core of the context-aware application. Sensors, either hardware or software, generate raw data that is presented by the context middleware to context-aware business logic. This process includes any controls on privacy mandated externally or by the application. The notification component provides the application developer with convenient, context-based facilities for communicating application information to people. Other business logic may be present, including components that interact with applications and data in other tiers. We also show a database management system in the figure, since effective use of context information may require access to more static, conventional data.

Although the architecture is generic, we used specific IBM products and research prototypes for the portal infrastructure and the context-awareness, privacy, and notification middleware. For definiteness, the description that follows includes details of the particular components we used.

This architecture exploits a number of mechanisms available directly in enterprise servers such as WebSphere [5] and WebSphere Portal Server (WPS) [6]. For example, we assumed that an enterprise directory, connected to the web server, would allow us to handle all issues of identification, authentication, authorization, and encryption with standard mechanisms in WPS. WebSphere provides a fully managed application environment for Java servlets and enterprise Java beans, including transaction management and database access. It also has transcoding and device-specific functionality that would be applied to the multiplicity of devices used to interact

with the system. Because we use browser technology, no other client software would need to be deployed on the devices.

The components for context acquisition and analysis (Context Weaver [1]), context-based notification (INS [7]), and policy-based secure access to sensitive context information (CPE [8]) are all recent IBM products or prototypes. We describe each of them briefly.

Context Weaver is a platform for writing context-aware applications, and is extensible, scalable, and reusable. It separates the concerns of accessing pervasive data from the concerns of aggregating and analyzing it. All sources of context information registered with a Context Weaver installation provide data to applications through a simple, uniform, XML-based interface. Applications access data sources not by naming particular providers of the data, but by describing the kind of data they need, and Context Weaver searches for an available source of such data. If the source fails, Context Weaver automatically rebinds the application to another provider of the same kind of data, if there is one.

Providers of context information include not only devices, services, and databases external to Context Weaver, but also programmed entities called composers, which compose context information from other data providers. Context Weaver applications use composers and external sources of context information interchangeably. The behavior of a composer is specified in a nonprocedural manner, using building blocks that interact with both passive and active providers of input context information. The nonprocedural approach frees the application developer from timing concerns, freeing her to concentrate on the logical mapping from lower-level context information to higher-level composed information.

The primary function of the Intelligent Notification System (INS) is to deliver a message to users in the most convenient fashion. The Notification Manager component of INS accomplishes this using preference data describing which devices to use, at what times, for which users. We envision the use of current context information both to deliver messages to users fulfilling a particular role, and to determine the best medium for delivering a message to a particular recipient.

We recognized that you could not responsibly build or support context-aware applications without also controlling access to the information. Many solutions we saw either ignored the privacy issue or built systems to avoid it (making

the system significantly less useful). Instead, we chose to address the problem directly.

The goal of the Context Privacy Engine (CPE) is to enable control of access to personal context information, through the specification of policies. Access can be managed through the use of groups, reducing the number of policies required. CPE also supports multi-level policies defining defaults that user preferences can override, except when enterprise policies take precedence. For example, if an enterprise outfits their vehicles with location-reporting devices, they can create defaults that allow all other employees to view the location information, the drivers can override those defaults to reflect their own wishes, but the enterprise can require that the dispatcher be allowed to access the location information regardless of the drivers' wishes. Architecturally, Context Weaver and CPE are independent components that can be configured to be aware of and leverage each other.

A final component publishes dynamic context information into a portlet page. In the normal case of a web page (including a portal/portlet page), any changes to the information on the page after it is rendered initially can only result from some separate thread of execution running in the browser at the client. In the case of context information originating and changing at some distant source, we need some general mechanism that can subscribe to the changing values, and block awaiting their arrival. One could imagine a Java applet issuing the requisite Context Weaver queries, and then rendering the resulting XML dynamically, once on initial page load, and as context changes arrived. The applet would execute in the browser, in the box labeled "Dynamic Content" in Fig. 1.

V. Demonstration Scenario

Before describing what we were able to prototype, we describe a fuller implementation, concentrating in particular on the sensors, adapters, composers, and context middleware. We need location sensors for people and medical carts, and medical devices (including call buttons) producing digital sensor readings. Adapters would need to be written for each, as would composers for higher-level context such as the co-location of a doctor and a patient or the last three patients seen by a doctor.

Our early experimentation with radio frequency identification (RFID) tags [9] suggests current products would be sufficient for locating people and equipment, assuming each patient room has a reader, and that there are other readers deployed throughout the hospital infrastructure. Patients have RFID tags as bracelets, medical staff have RFID tags as badges, and carts have tags attached to them. The adapter for the readers associates each reader with a hospital location, and a particular tag with a patient, doctor, nurse, or cart. Tag events are converted by the adapter to XML data giving hospital locations of individuals and carts.

Adapters for medical devices convert telemetry to an XML-based format that includes a unique device identifier. Device identifiers are associated with patients through some hospital business process that depends on the device characteristics and hospital policy. RFID tags and the co-location of devices and patients can be used in implementing a particular process.

Above the device level, composers filter and aggregate data. Doctor-location events trigger the display of a particular floor map in the doctor's portal, populated with visual indications of any patients located on the same floor. A "patient-visit" composer uses a doctor's list of patients, the patients' locations, and the doctor's location to generate a context event whenever the location of the doctor matches that of one of the patients. Business processes that subscribe to those context events use patient-visits for billing, presenting patient information to the doctor, or notifying Joann of Dr. Able's visit to John. A very simple composer tracks the last three patient-visit events for the doctor's portlet. A slightly more complicated one computes the three closest patient locations.

Some context information is acquired from software such as instant-messaging applications or operating systems. We have an adapter for Lotus Sametime, IBM's instant-message suite, that provides conventional "Active," "Do Not Disturb," or "Unavailable" status, and indications of what application is active. Composers can use this and other information (such as calendar entries) to compute an indication of whether a doctor can be interrupted. In our scenario, the notification infrastructure refuses to deliver notifications during patient visits unless they are critical.

Clearly, the scenario required significant pruning for the purposes of building a modest demonstration. Nevertheless, we wanted to build a demo that was architecturally realistic and true to the vision of the scenario. To that end, we built a simple portal application with WebSphere Portal Server and Context Weaver, rather than including all the components of Fig. 1.

We decided to concentrate on what we came to call the nurse triage portlet, and a patient portlet that provides additional information. The triage portlet provides a nurse with a list of the patients in his/her ward, sorted by the seriousness of current alerts. The alerts are context information, originating in various devices that monitor each patient, such as blood pressure, heart rate, or body temperature. This raw telemetry for a patient is processed by a Context Weaver composer that encapsulates medical knowledge for evaluating the seriousness of the combined patient context, and generating appropriate alerts for the nurse. Fig. 2 shows the run-time structure of the system for a nurse with two patients.

At the top of the figure, the two portlets are implemented as in Fig. 1: the user accesses them through a client web browser, the portlets are rendered by WebSphere via Java Server Pages (JSPs), and the underlying portal application executes in WebSphere Application Server, with external connections to a Context Weaver server and a relational database. The "business logic" obtains information from the database about patients and nurses, while the "context-aware business logic" is implemented in Context Weaver composers that provide patient alerts and telemetry.

At the bottom of the figure, patient monitors feed raw telemetry into Context Weaver, where it is merged by the
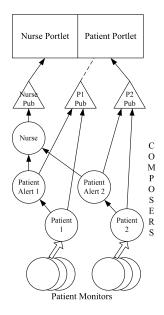
Fig. 2.   Demonstration Healthcare Application

patient composers into one source per patient. This source is analyzed by a second level of composers to identify alerts, which, in turn, are fed to the nurse composer. This data is then published to individual Java Messaging Service (JMS) "topics" and rendered in the portlets. The dotted line from the middle publisher indicates that the patient portlet is currently displaying information from Patient 2 rather than Patient 1.

Alerts are presented on the user's display in real time. In the triage portlet, the arrival of an alert reorders the list of patients and alerts dynamically. Each patient has an associated button that can be pushed to cause the patient portlet to display more information. The patient portlet also changes dynamically in response to alerts or underlying telemetry changes that may not be significant enough to cause an alert.

Fig. 2 glosses over issues that are architecturally significant. For example, we had neither real patients nor real medical devices. We wrote simple Java applications with a user interface to control the streams of telemetry from the simulated monitors, and the associated adapters to produce the XML data from the telemetry. We were also unable to include INS and CPE in the prototype, due to lack of human resources.

We used JDBC links with a simple database to augment the raw context information with more static information: the database implemented a simple model of nurses, their patients, patient biographic information, monitors, patient medication, and so on. This was an important demonstration point, since realistic context systems will of necessity rely on additional information external to them. The database was used both by Context Weaver (composers, adapters, and related elements) and by the portlet applications themselves.

Using dynamic HTML to maintain real-time information in the portlet pages is itself an interesting exercise. The most direct solution would be to use a Java applet that implemented a Context Weaver client. Since the existing Java client was not engineered with applets in mind, we chose a different approach that had the advantage of reusing a small "web messaging" applet on the portlet page. The applet subscribed to the JMS topics from the publishers, but this did not deal with the initial values displayed on the page. Waiting for all patients to produce an alert or a clear indication might take minutes or even hours. While Context Weaver data providers make their current values available on request, JMS topics have no such feature, forcing us to a second method for acquiring and displaying the initial values of patient data. The servlets query Context Weaver for the initial XML data, and place it in a string constant in the page source, so that, in the browser, it can be parsed and displayed with the same JavaScript code as each JMS message received from a publisher. Although this reuses at least some of the code for initial and ongoing rendering, the initialization is very awkward and inelegant.

VI.   WHAT HAVE WE LEARNED?

Although our demonstration prototype was very modest, it taught us a number of lessons about pervasive systems and their use in an enterprise context.

In our prototype, we chose to transform the Context Weaver data from XML into dynamic HTML at the browser. We used browser methods to parse the XML into a DOM (document object model) object, and JavaScript to combine it with the DOM tree of the page. This approach could be generalized in several ways. One would be to replace the JavaScript functions with a call to a browser method that applies an XML-stylesheet transformation to it. However, those facilities are not yet standardized. Also, if a different context middleware infrastructure were used that did not produce XML-based data, this approach would fail. We are now seeking more general approaches to convenient dynamic inclusion of context information into web (or portal) pages. We expect this to result in a number of Context Weaver tools, including one that uses XSL or other transforms in the second tier to minimize the computation in the browser. Those tools would presumably be general enough to apply transformations to non-XML data.

Mounting a convincing proof of concept, especially in the medical domain, is a daunting undertaking. At a technical level, the architecture of Fig. 1 includes many complex technologies that need to be assimilated by the designers and developers of the complete system; it is hardly an exaggeration to say that each tier and component of the architecture brings its own, significant, learning curve. There are plausible features of our demonstration that we did not even attempt to prototype: acquiring actual medical devices and writing Context Weaver adapters for them, obtaining medical expertise for plausible composers for the patient and nurse alerts, understanding HIPAA well enough to include CPE in a meaningful way, or attempting to use an actual fourth-tier medical application.

The generality necessary to incorporate pervasive-computing applications into an enterprise setting exacts a cost in terms of performance and complexity. In spite of the power of modern desktops, laptops, PDAs, and servers, responsive pervasive-computing applications will still require significant

attention to performance issues for the foreseeable future. This is likely to remain true as open standards continue to develop. While they indeed enable interoperability, they always push the edge in terms of performance and responsiveness.

Perhaps the most important lesson we learned, however, is related to the overall integration issues we hope our scenario has illuminated. For enterprise pervasive computing to become a reality in a hospital, significant issues have to be faced in identifying and integrating role-based authentication, not only across the enterprise, but "down" into even the small devices and applications of pervasive computing. Our intuition suggests the same would be true in other enterprises. Other integration points such as databases and enterprise applications also add code and complexity, putting pressure on how light-weight the applications can in fact be. Designing pervasive-computing applications for this enterprise environment will remain a challenge for some time to come.

VII. CONCLUSIONS AND OUTSTANDING ISSUES

Enterprise smart spaces like the healthcare scenario we consider can only become common with the emergence of an "ecology" of pervasive computing that runs the gamut from device adapters for raw context to packaged applications that both observe their context and take action in it. To date, our efforts have been concentrated on context awareness, and so the outstanding issues we have identified tend to be motivated by Context Weaver and related components.

A context service like Context Weaver can amortize the cost of incorporating new sources of context across many context-aware applications and services, and provide application developers with a common API for context information. This can be the core of an ecology, where standardization of context data and access to it provide the foundation for many context-aware pervasive applications.

Despite couching our discussion in terms of Context Weaver, INS, and CPE, we believe strongly in the general architectural importance of middleware. For pervasive computing to become common, there need to be standards, conventions, interfaces, protocols, and tools that foster the participation of a wide range of system components in the pervasive environment. In turn, this systematization enables middleware to provide value by insulating components from proprietary interfaces and providing layers of abstraction to ease the developer's task.

Although we do not discuss them in this paper, the standard issues in mobile computing of effective mobility, intermittent connectivity, and fault tolerance remain important impediments to an enterprise placing reliance on small and mobile devices for mission-critical functions.

We mentioned issues of multiple user interfaces in the scenario, and whether the working environment of the healthcare professionals would be so fragmented across devices and applications that it would be unwieldy. This requires careful, system-level attention to computer-human interaction issues, and may prove quite a significant barrier.

Finally, pervasive computing scenarios and prototypes permit few conclusions to be drawn about how these applications might scale in an enterprise setting, with many users, applications, devices, interfaces, and communication media. However, our architecture builds upon scalable enterprise solutions like WebSphere application and portal servers, and relational enterprise database technology (DB2, in our case). The prototype software, including Context Weaver, INS, and CPE, has been designed with enterprise requirements in mind, and so we believe the application architecture has the potential to scale to the size suggested by our hospital scenario. Real experience would be extremely valuable, but also seems unlikely in the near future.

REFERENCES

[1] H. Lei, D. M. Sow, J. S. Davis II, G. Banavar, and M. R. Ebling, "The design and applications of a context service," *Mobile Computing and Communications Review*, vol. 6, no. 4, pp. 45–55, 2001.

[2] N. H. Cohen, H. Lei, P. Castro, J. S. Davis II, and A. Purakayastha, "Composing pervasive data using iQL," in *Proc. 4th IEEE Workshop on Mobile Computing Systems and Applications (WMCSA 2002)*. IEEE, June 2002, pp. 94–104.

[3] N. H. Cohen, A. Purakayastha, L. Wong, and D. L. Yeh, "iQueue: A pervasive data composition framework," in *3rd International Conference on Mobile Data Management (MDM 2002)*. Singapore: IEEE, January 8–11 2002, pp. 146–153.

[4] A. Crabtree, T. Rodden, T. Hemmings, and S. Benford, "Social aspects of using large public interactive displays for collaboration," in *Proc. Ubi-Comp 2002, the 4th International Conference on Ubiquitous Computing*, ser. LNCS, no. 2498. Springer-Verlag, September 29–October 2 2002, pp. 229–236.

[5] WebSphere software platform. IBM. [Online]. Available: http://www.ibm.com/websphere

[6] WebSphere Portal for Multiplatforms. IBM. [Online]. Available: http://www.ibm.com/software/genservers/portal/

[7] V. Bazinette, N. H. Cohen, M. R. Ebling, G. D. H. Hunt, H. Lei, A. Purakayastha, G. Stewart, L. Wong, and D. L. Yeh, "An intelligent notification system," IBM Research Division, Thomas J. Watson Research Center, P. O. Box 704, Yorktown Heights, NY 10598, IBM Research Report RC 22089 (99042), June 2001.

[8] M. Blount, J. S. Davis II, M. R. Ebling, W. Jerome, B. Leiba, X. Liu, and A. Misra, "Context privacy engine: A system for managing privacy in context-aware computing environments," March 2004, submitted to Middleware 2004, 14pp. in ms.

[9] RFID.org home page. Association for Automatic Identification and Mobility (AIM). [Online]. Available: http://www.aimglobal.org/technologies/rfid/