Impress: Towards Next-Generation Ubiquitous and Pervasive Computing Systems[*]

James P. Black Hao Chen
Omar Zia Khan
David R. Cheriton School of Computer Science
University of Waterloo, Waterloo, Ontario, Canada. N2L 3G1
`{jpblack,h8chen,ozkhan}@uwaterloo.ca`

October 3, 2007

Abstract

Research in pervasive and ubiquitous computing has resulted in a number of prototype systems developed and implemented in isolation. If this research area is to have impact, the next generation of research prototypes must move beyond single laboratories and isolated, independent software frameworks. Future systems must not only inter-work and inter-operate, but should enable more rapid prototyping and incorporation of components and software artifacts developed by others. Developing and exchanging software, including the concepts embodied in it, requires an approach based on standards and widely available software infrastructure.

The main contribution of the paper is to show that an appropriate infrastructure for next-generation ubiquitous and pervasive computing systems can be found in the Jabber open-source community. Jabber, now standardized as XMPP, was originally conceived as an open-source alternative to proprietary instant-messaging systems. Its design, architecture, and implementation provide a rich, extensible, standardized selection of concepts, protocols, servers, components, and clients. We show how Jabber provides an excellent foundation for the implementation of smart spaces, and a robust platform on which to build a ubiquitous-computing ecology, while supporting ambitious research goals in second-generation smart spaces. We also discuss various proof-of-concept prototypes that we have developed to validate our argument. Finally, we demonstrate how Jabber has allowed us to pursue research in pervasive computing without requiring substantial effort in a start-up phase.

[*]This research was supported by a Discovery Grant of the Natural Sciences and Engineering Research Council of Canada.

1 Introduction

For twenty or more years, researchers have been pursuing a vision of ubiquitous (or pervasive) computing ("ubicomp"), articulated eloquently by Weiser [19]. We, in the research community, agree on the vision, by and large, and it is clear that progress in electronic and computer technology makes the vision appear ever closer to reality: we have smart phones and personal digital assistants (PDAs), various wireless technologies, ever-faster processors, larger memories, larger displays, faster networks, and digital media streaming all around us. A significant remaining challenge is moving beyond first-generation projects in ubiquitous computing to systems, middleware, applications, and devices that can interwork, enable an "ecology" of ubicomp, and lead to effective deployment and commercialization.

We begin by reviewing a representative sample of first-generation ubicomp systems in the next section, to identify recurring concepts and approaches, and describe a general model of a ubicomp system. Based on this model, Section 3 discusses the set of abstractions needed for next-generation ubicomp systems. In Section 4, we elaborate our main argument, which is that the Jabber open-source community already provides a sound basis for many of these abstractions, while also providing a strong foundation for future research, without requiring the development of yet more project-specific bases on which the research can progress. In Section 5, we describe a number of proof-of-concept prototypes we have built based on Jabber, as part of the Impress project, followed in Section 6 with a brief discussion of research activities we are pursuing. Section 7 provides our conclusions.

2 Previous Work

There have been a number of large ubicomp projects over the last decade, each with particular goals and emphases. We provide some details on four of them, Gaia, Aura, Cooltown, and Interactive Workspaces, as representatives of a much larger body of work. Our objective is to identify recurring themes in these projects that can guide the development of future systems.

2.1 First-Generation Projects

The Gaia project [16] introduces the term "active space" to capture the concept of a smart environment in which users are followed by their data. Gaia components and applications are hosted by a Component Management Core that manages them dynamically. The basic system services include context, presence, a space repository, an event manager, and a context file system. Gaia is built using CORBA and leverages its built-in components for functionality such as the event manager, which is essential for communication. The presence service tracks the existence of entities while the context service handles other types of context. While these services report the presence and context in real time, the space repository stores this information for possible future reference. The context file system allows data to be associated with a context, and then made available within it.

Aura [9, 17] is built upon Coda, which provides disconnected file operation, Odyssey, which provides application-aware adaptation, and Prism, which is responsible for composing high-level tasks to enable proactivity. Tasks are treated as first-class entities that have platform-independent representations. This allows task execution to include the use of heterogeneous service providers from different platforms. Prism comprises a context observer, an environment manager, suppliers, and a task manager. The context observer reports changes in context, the environment manager enables service and device discovery, and suppliers provide the abstract services to the environment manager. The task manager is responsible for task migration when users move. User intent prediction is concerned with determining whether the user requires a task at a particular time based on the output of the context observer.

The Cooltown project [13] uses the notion of web presence for people, places, and things. Its goal is to build a web-based system for nomadic users. Each object has an associated URL that provides further information regarding it, and web servers are placed in devices and associated with real-word objects. Different devices can exchange content either by sending information directly or through the transmission of a URL. This communication is initiated by the user, since there is no concept of context awareness in Cooltown. Once the user initiates the process, service discovery is simplified and the interfaces to different devices and services are exposed via their URLs without requiring any other software.

The Interactive Workspaces project [10], with iRoom as its experimental research environment, focuses on control management, data movement, and dynamic application coordination. The project uses a tuple-space model for storing distributed events using an event heap. A data heap is used for storing and locating data belonging to different applications, with the ability to convert data in different formats for exchange. iCrafter, the service framework for this project, is responsible for service invocation and dynamic interface generation. iCrafter returns an interface for a service based on the status of the event heap. The project is not that concerned with sensing context and acting based on it, but concentrates on enabling collaboration among applications that are now aware of each other.

2.2 Recurring Themes

Despite the differing viewpoints, terminology, and goals of these systems, there are a number of common issues or themes that can be discerned. Generally, the "system" consists of one or more "smart spaces" (or a similar term), where users, devices, software, and hardware interact. All projects rely on middleware to provide communication and coordination among the cooperating entities, and the middleware is typically designed as multiple related layers and/or components. Common middleware functions include event distribution, data storage, service and resource discovery, user identification, authentication, security, and privacy, and context modelling and representation.

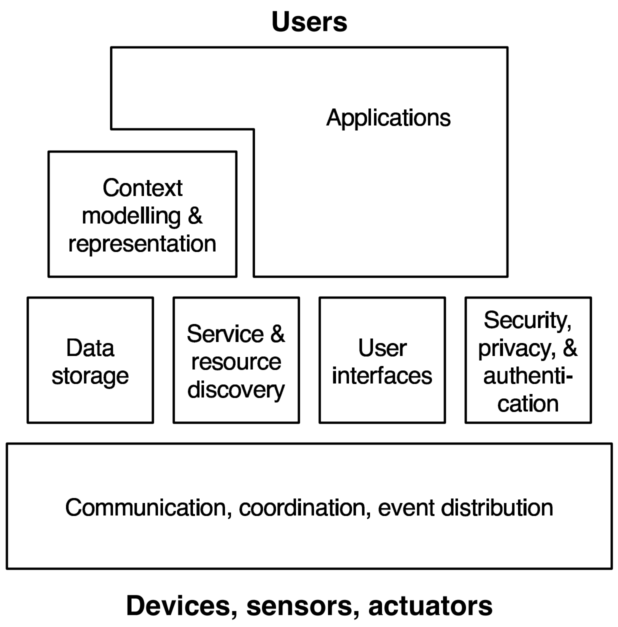

Users
Applications
Context modelling & representation
Data storage
Service & resource discovery
User interfaces
Security, privacy, & authenti- cation
Communication, coordination, event distribution
Devices, sensors, actuators

Figure 1: Generic architecture of a smart pace

These components or functions cooperate to link the devices, sensors, and actuators in the smart space to applications and actual users. See Fig. 1.

The smart space itself usually corresponds to a single physical space such as a room, building, or courtyard. It often contains physical sensors, devices, and actuators, under the control of the user, the applications, or the system itself. Some of the devices are assumed to move with users as they come and go, although some may be fixed in the space. The applications are what make the smart space more than just a collection of devices and technologies, as they usually embody design principles or goals such as "calls following people," if communication sessions are to be re-routed to a different space when or as the user moves. One might argue that the boundary between the applications and the infrastructure is not always clear; this reflects the immature state of the art, in our view.

Communication and coordination facilities can be enabled through RPC systems, through distributed object technologies such as CORBA, or otherwise using a message-oriented middleware. Some researchers, such as Ranganathan and McFaddin [15], make use of current internet technologies, including web services.

Since ubicomp environments are supposed to react to real-world events as they occur, event publication and distribution are usually handled by a particular system component. Often, events are generated or collected regardless of whether any particular entity is actively interested in them, leading to an intentional decoupling between sources (or publishers) of events and those entities interested in being notified (the subscribers).

The decoupling leads to cleaner system structure, since publishers and subscribers need not be aware of each others' identities. There seems to be growing agreement in the research community on the use of some sort of publish-subscribe ("pubsub") system.

Data for users and applications needs to be stored and made available at different types of devices. Data storage may be implemented by conventional (distributed) file systems, but does not usually have the "public" nature of context information, nor would it necessarily be subject to the same type of standardization.

Most ubicomp systems assume that users move among smart spaces at will, appearing and disappearing from time to time. The same is often assumed about services and other resources (such as portable devices), and so discovery of entities is an important issue both for a smart space and its users. This leads to some sort of discovery protocol (typically called service discovery) which ensures that different entities such as users, devices, and applications can be located. The service-discovery protocol needs to respond to queries in a flexible manner, since users in an unfamiliar environment may not have enough information to construct very precise queries.

Security in pervasive computing comprises two essential factors. First, the system should authenticate the user properly. Second, the system should enforce access control policies so that privacy of data is not compromised. This requires granting different levels of access to different types of users in different situations.

Users in a smart space expect to interact with it through a variety of devices, such as cell phones, PDAs, full computer workstations, touch screens, and specialized devices like RFID tags and readers. Different user interfaces must be supported so that the user experience can be as natural as possible.

Many ubicomp papers speak of "context" as information that can be used by people, applications, and devices to enable or improve the way the systems responds to users and events. A commonly-used definition from Dey [6] is: "Context is any information that can be used to characterise the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and applications themselves." While intuitive, the definition is neither technical nor constructive. One of the most common items of context information is often called "location," meaning the logical or geographic place where a service, device, or person happens to be. Another common example is some representation of the activity in which a person is engaged.

In most projects, the context information has several important properties.

- Context information is available to many applications, entities, or people, subject to access control.
- Individual context consumers are generally interested only in some of the context information available to them.
- Context information is produced by a variety of sensors and/or applications, in differing forms, and its semantics and representation are subject to standardization.

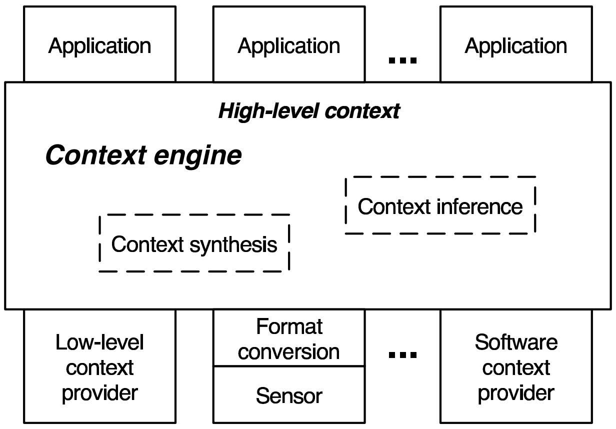


Figure 2: Typical architecture of a context engine

- The context information may vary significantly from one smart space to another.
- Different types of context information may change on dramatically different time scales, from milliseconds to days.
- Context information may be processed (filtered, analyzed, correlated, combined, ...) to produce higher-level or more abstract context.

In the rest of this paper, when we speak of context or context information, we mean information available in standardized forms to interested entities.

One difference between pervasive computing and its predecessors is that it involves human activities and social situations, such as a presentation or a health-care scenario, where events happen dynamically and unpredictably. To be able to represent, reason about, and react properly to context becomes a basic requirement for effective ubicomp. At an abstract level, a number of projects including [4] and [11] have adopted an architecture like that of Fig. 2. Raw information from simple or complex sensors, or from software sources, forms the basis for context. The information must be represented in or converted to some standard form, which can be made available to applications though some query interface. Often, the context engine itself allows abstract context to be synthesized or inferred from other primitive context. The key point is that applications are able to query the context in which they operate through some third party (the context engine), which provides independence and decoupling of the application from the lower-level details of the specific context available in a given situation. However, this research is still far from mature. In addition to the problem of poorly inferring high-level context from low-level information, differing representations and semantics of context prevent any real collaboration or integration among smart-space frameworks from different project. Realizing this problem, researchers have begun to use ontologies and semantic-web tools to represent and reason about context.

At the highest level in Fig. 1, applications are written to make use of smart-space facilities, services, and context. However, the architecture has not yet solidified to the point where there is are clear distinctions between an application, a system utility or feature, and an infrastructure mechanism.

3 Requirements for the Next Generation

As a result of the many first-generation ubicomp research projects, a general consensus has emerged on the central abstractions of the field: smart spaces, context awareness, and lower-level system components that are expected to form the basis on which applications and system utilities are built.

However, the consensus is still too embryonic to provide an agreed set of standards to which to build any of the elements of Fig.1. Without those standards, we cannot exchange applications, software, or components; we cannot easily incorporate a wide variety of user interfaces, sensors, and actuators; and we cannot free our software from detailed and constraining assumptions about how to interact with people, services, applications, and physical smart spaces. Unless we can create this kind of ubicomp "ecology," we will be unable to progress beyond yet more isolated projects with software and systems that fail to emerge from the lab in which they are crafted.

Before our second-generation ubicomp research projects can have impact, we, as a community, need to:

1. Agree on a set of standards and interfaces to which we design and build our software, and wrap our devices, sensors, and actuators;
2. Ensure a "low barrier to entry" of the ecology, so that many researchers, software and hardware developers, vendors, and users can participate; and
3. Agree, in particular, on a technical model for representing people that is good enough for practical systems, but that can be changed, extended, or replaced as better understanding and consensus emerge.

The standards we have in mind would provide a reasonably rich semantic basis for smart spaces. They are no longer research issues in their own right; we simply need to choose a practical, coherent framework that is rich enough to support future work. The framework needs to include the following.

- A naming model for people, systems, services, applications, and devices.
- A model for user security and privacy.
- A model for discovering devices, services, users, and other entities in a smart space.

- Client, component, and server software and hardware platforms.
- A data store, tuple space, or publish-subscribe model for information and event storage and dissemination.
- A user-interface model and facilities for interaction with components, services, applications, and other users.

If the standards are open and supported by an effective open-source community, the barriers to entry into the ecology can be greatly reduced, not only for researchers, but for "real" participants as well.

Some work has been done on standardizing context for ubicomp, such as SOUPA [5] (Standard Ontology for Ubiquitous and Pervasive Applications) and FOAF [3] (Friend Of A Friend) ontologies, for some ubicomp concepts and people, respectively. These are a start, but much more work needs to be done to agree on a more comprehensive set of context information.

For example, in addition to conventional "biographic" information such as identifier, name, and address, we need a variety of other context about people like their current physical location, entries on their calendar, and preferences for "buddies" that can use personal context. We think of our extended model of people as consisting of a basic, biographic core like FOAF, supplemented with an extensible set of other context, as shown in Fig. 3. In the figure, "Tune" refers to the piece of music the user might be listening to, "Mood" to his/her self-reported mood, and "ACL" stands for the various access-control lists the user might have created for personal information.

In fact, we argue that the next generation of ubicomp systems should be based on semantics, standards, and software from the world of instant messaging, and, specifically, Jabber. Users should communicate with their smart spaces like they communicate with their real-word buddies. In the next section, we describe why Jabber is a logical choice.

4 Jabber for the Next Generation of Pervasive-Computing Systems

Above, we discuss various abstractions for pervasive computing. In this section, we present Jabber, an instant messaging and presence system, and analyze its suitability as an underlying platform for pervasive computing. Jabber provides a unified suite of concepts, protocols, and software presents a good starting point for building the next-generation pervasive and ubiquitous computing system.

4.1 Jabber and its Architecture

Jabber (`www.jabber.org`), based on the Extensible Messaging and Presence Protocol (XMPP), is a popular open-source system that enables the exchange of messages

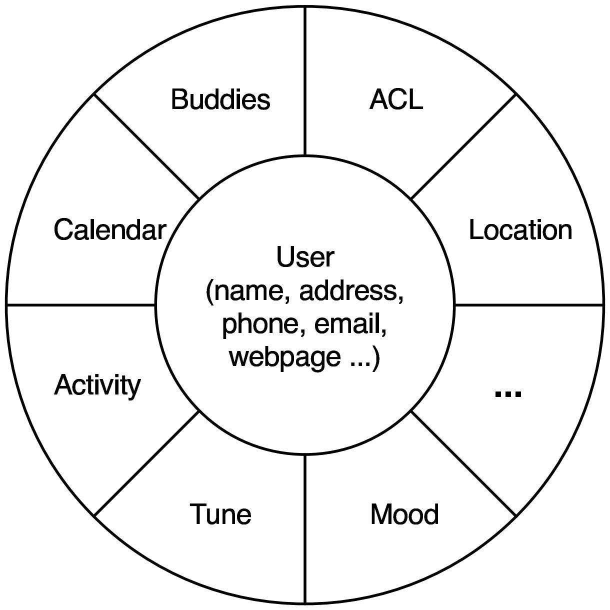


Figure 3: User model

and presence information in near real time. Jabber is inherently distributed, with many Jabber servers acting as peers that enable communication among the users served by each of them. This architecture is analogous to that of email systems. As shown in Figure 4, each server communicates with client devices, client applications, and other servers; some of the servers also support gateway functionality to other messaging systems. The connections among the Jabber servers are created and destroyed as needed.

Clients and applications connect to a Jabber server over a secure connection, as specified in the XMPP core [2]. The basic instant-messaging and presence functionality of XMPP is described by [1], which provides features very similar to proprietary applications like Microsoft's Windows Live Messenger, Yahoo! Messenger, AIM, or IBM's Lotus Sametime. Jabber has recently gained important momentum since being adopted as the core technology behind Google Talk. Google's service and that of a number of private Jabber installations provide ample evidence of its scalability and robustness.

As a platform for instant messaging and presence, there are good Jabber clients with familiar interfaces for almost any computing platform, from Windows, Macintosh, and Linux through Java applets to Palm PDAs, PocketPCs, and cell phones. The Jabber server architecture is itself distributed, and gateway software is available for communication with other IMP systems. In addition, there are a number of multi-protocol clients that simultaneously communicate over Jabber and other IMP protocols. This core Jabber functionality provides excellent communication facilities not only among people, but between people and applications or services. It includes a naming model, called Jabber IDs, or JIDs, where names look like <user>@<server>/<resource>, although

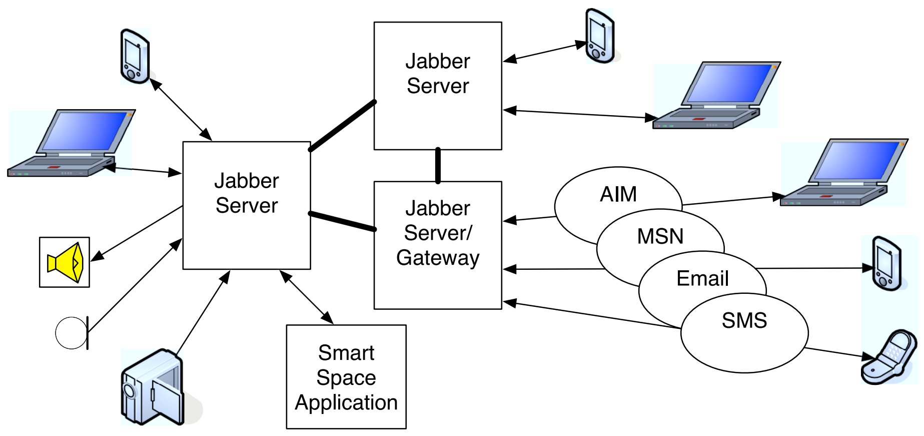


Figure 4: Jabber Architecture

only the <server> component is required. Typically, JIDs look like email addresses.

The Jabber model for user security and privacy is relatively sophisticated compared to other IMP systems. Client-server communication is normally encrypted, and users can control not only what presence information is released about them to "buddies," but also whether (non-presence) messages can be received from other parties. In addition, when desired, end-to-end communication via Jabber can also be secured.

Jabber has a simple yet functional protocol for discovering features and services implemented by an entity, and this can be applied directly to discovering devices and other components of a smart space.

Jabber supports a component model by which other services can be associated with an existing Jabber server. This is used by a number of components, such as multi-user chat, and publish-subscribe. The multi-user chat component provides a fairly general facility for group communication, and may well provide interesting mechanisms for modeling, say, all the devices and users in a single physical room. The publish-subscribe protocol provides a very general data storage facility that can store data persistently, or simply distribute it to subscribers. Published data can be arbitrary, and subscribers can either request particular items, or have them forwarded as they arrive.

The exchange of presence information is controlled by "subscriptions." To subscribe to a user's presence, an explicit request must be granted by the user, and subscription is typically (but not necessarily) symmetric between the subscriber and user. A user's subscriptions are stored by the server in his or her "roster," along with other JIDs the user wishes to preserve. The exchange of (non-presence) messages with a user is controlled by a "privacy list," which is stored by the server, and maintained by the user.

Jabber is based on the use of XML "stanzas," of which there are three types at the top level: <presence />, <message />, and <iq />. Additional functionality is standardized through the XMPP Extension Proposal (XEP) process. Typically, each XEP results in one or more new stanzas defined within the <iq /> hierarchy. The discovery protocol ("disco") is [XEP-0030].[1]

Having introduced the Jabber architecture, we now examine the support Jabber provides for the recurring themes of Section 2 and the abstractions outlined in the previous section.

4.2 Support for Recurring Themes

Jabber provides a certain level of support for all of the recurring themes listed earlier. The core Jabber functionality provides excellent support for communication among different entities. These entities can include human beings, devices, and applications. This allows using Jabber to be used as message-oriented middleware for communication and coordination within a pervasive-computing environment.

Jabber provides a complete pubsub framework [XEP-0060] to allow decoupling between posting and receiving information. Each pubsub server or component provides any number of named "nodes," and users can be subscribed to those nodes if authorized by the node owner.

In addition to pubsub, Jabber allows users to store data in XML form on the Jabber server [XEP-0049], although the pubsub [XEP-0060] provides more generality, and both may not be needed. This data storage facility can be used to store user data and retrieve it as and when needed. This XML storage is not suitable when the user data needs to be shared with other entities.

Disco [XEP-0030] allows hierarchical discovery of resources, components, and services provided at a given JID (*i.e.,* at a server). It is possible to extend this protocol to implement a more powerful mechanism to satisfy the requirements of a pervasive computing environment with respect to service discovery.

The security model of Jabber (as embodied in XMPP core, XMPP IMP, and the XEPs) can cater to conventional requirements for security in pervasive computing. Both client-to-server and server-to-server communications can be encrypted. Users can also control which entities are allowed to subscribe to their presence information. They can tell the system to block messages from any other entity. Similarly, pubsub node owners can specify the roles for different entities with respect to publishing and receiving at each node. For all these cases, Jabber can be configured to require explicit authorization before any request to subscribe to a node or user's information is approved.

Since Jabber is primarily an IMP system, Jabber clients are used for user interaction. It is possible to extend the Jabber clients for more structured communication using the Data Forms protocol [XEP-0004]. This provides the option of developing new interfaces for interaction with users.

[1]The complete set of XEPs is available at `www.xmpp.org/extensions/`.

4.3 Support for Abstractions

Jabber also provides for most of the abstraction listed in the previous section. An enhanced user model can be adopted easily through various Jabber protocols that support storage of user-related information. These protocols include User Mood [XEP-0107], User Activity [XEP-0108], User Physical Location [XEP-0112], User Geolocation [XEP-0080], User Tune [XEP-0112], User Avatar [XEP-0084], and User Profile [XEP-0154]. Apart from these specific protocols, the Personal Eventing via Pubsub protocol [XEP-0163] allows the publication of specific information related to a user for distribution via pubsub. Thus Jabber not only provides a rich notion of a user, it also allows the user to control access so that the information is only distributed to the desired entities. The functionality suits the requirement for a rich user model for pervasive computing.

As mentioned above, the core Jabber protocols are based on IETF RFCs, and all extensions are also standardized through the XMPP extensions process. The extension protocols are initially presented in draft form, deliberated upon by the standards committee, and then voted into acceptance after all necessary changes. The extension process is managed by the XMPP Standards Foundation. Thus, any pervasive computing platform built using Jabber will also be based on standardized and open-source protocols.

Since Jabber is open-source, a strong community of developers has evolved that distributes various pieces of software written using the protocol. It is possible to download different implementations of the Jabber servers and clients with associated libraries. New users can download and reuse existing servers and clients. They do not need to implement the Jabber protocol from scratch, and instead can focus upon development specific to pervasive computing.

4.4 Alternatives

Throughout this section we presented Jabber as a platform for the next-generation pervasive-computing environment. Here we mention several other candidates that we considered and explain why we prefer Jabber to them in the Impress project.

SIMPLE (SIP for Instant Messaging and Presence Leveraging Extensions, `www.ietf.org/html.charters/simple-charter.html`) is also a messaging and presence system standardized by the IETF. It builds upon SIP and provides similar functionalities to Jabber IMP. We chose Jabber for several reasons. First, Jabber is a large open-source community with an impressive and vibrant composition. This allowed us to reuse the software provided by Jabber and concentrate on using it for pervasive computing rather than implementing the base protocol. SIMPLE, although an open standard, has not engendered the same kind of community, largely remaining a protocol implemented by proprietary commercial interests. Second, the extension protocols for Jabber and their obvious utility for pervasive computing are much less developed (or non-existent) in SIMPLE. This again relieves us of the responsibility to design and standardize all these protocols.. We simply use them for developing the functionality needed for pervasive computing. SIMPLE may be able to serve as a similar kind of platform, but the amount of work needed using Jabber appears much, much less.

The use of distributed object technologies, such as CORBA or Java RMI, is another possibility. We again considered Jabber more appropriate because it neither requires extensive software installation on the participating devices nor needs significant processing or memory resources. Due to its specialization for IMP, it provides many abstractions not appropriate for the lower-level semantics of CORBA and Java RMI. Web services have a similar problem, in that they are unwieldy for our purposes and lack many of the abstractions implemented by Jabber.

We find that only Jabber allows us to do rapid prototyping of pervasive computing systems. With any of the alternatives, we would have to design certain protocols to achieve desired functionality, and then implement most of the bits and pieces needed as infrastructure for pervasive computing. This Jabber-based approach not only conserves our scarce resources, but saves us from retracing steps already followed in first-generation projects.

5 Proofs of Concept

Over the last three years, we have used a variety of Jabber libraries, components, servers, and clients. Originally, we simply downloaded one of the open-source servers (Jabberd, at the time), and ran it in our laboratory with conventional instant-messaging clients like Psi (`psi-im.org`) and Exodus (`exodus.jabberstudio.org`). With no prior experience of any IMP software, this took only a few days of elapsed time, and only a few hours of total time. Since then, we have gained experience with other servers and components, and have written components, clients, and applications, all as Jabber entities.

X10 component. The first component we developed was one to wrap X10 devices as Jabber entities. X10 (`www.x10.com`) is a simple home automation system that controls devices by modulating a data signal onto the power lines in the home. See Fig. 5. Each of these devices appears to be an individual Jabber entity, with its own presence and buddy list. Other users can subscribe to these entities, so that when there is a presence change, such as an X10 lamp being turned off, subscribers receive a presence-change notification. Users can also send text commands to these entities to control them, such as turning on/off an X10 electronic fan. The component was written to use an open-source library for the Jabber component protocol [XEP-0114]. We are currently working to implement the ad-hoc command extension [XEP-0050], so that other entities can automatically discover what commands the devices accept, and how to invoke them.

Context browser. Initially, the only implementation of Jabber pubsub available as open source was Idavoll (`idavoll.jabberstudio.org`). Today, however, we

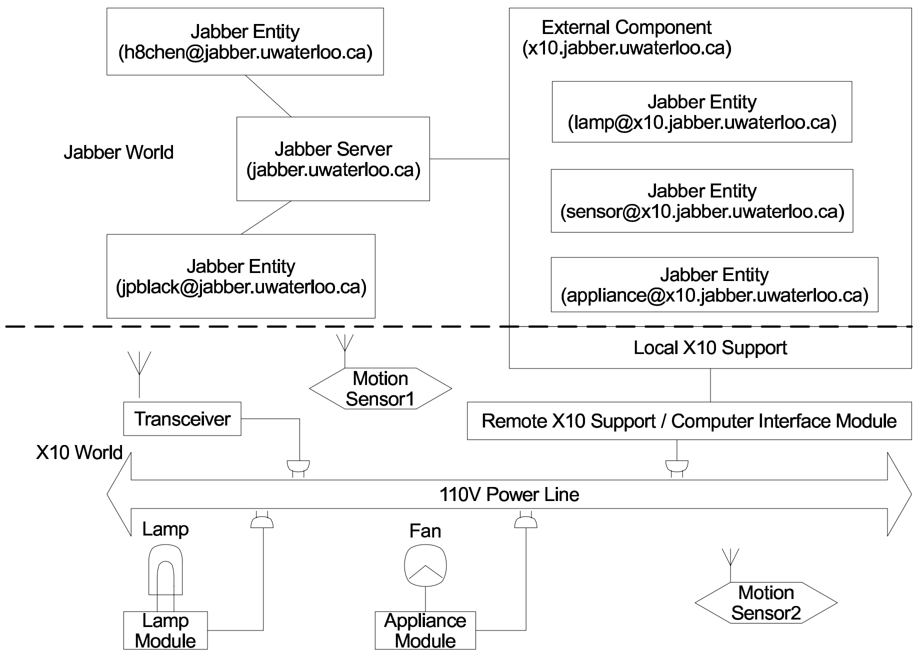
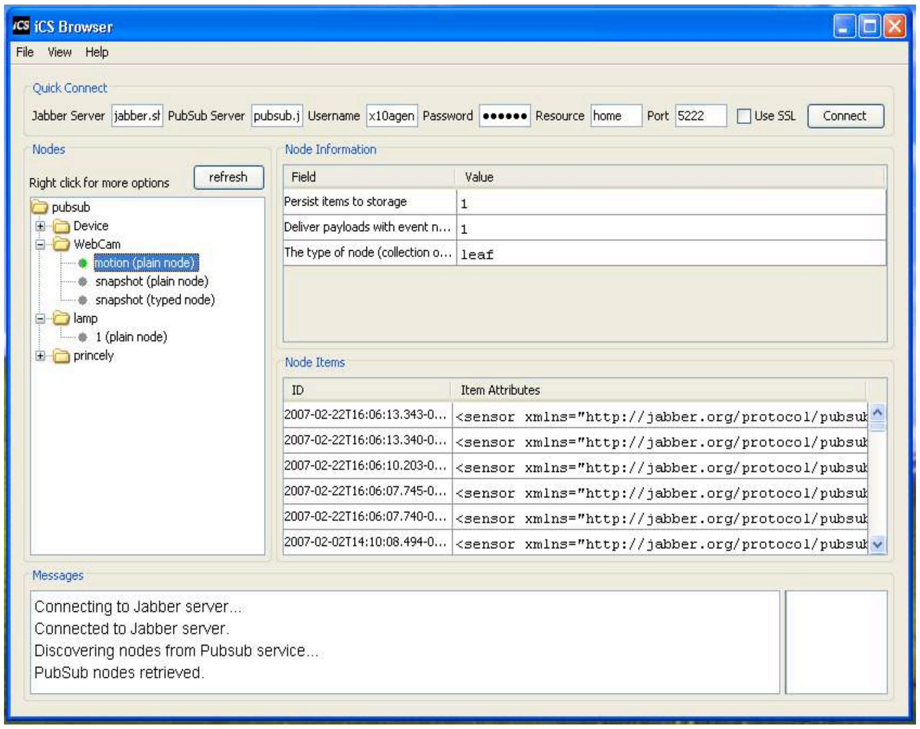


Figure 5: X10 and Jabber

use the built-in pubsub service of the open-source Openfire server (formerly Wildfire, `www.igniterealtime.org/projects/openfire`). Naturally, we wanted to use the pubsub service as the basis for our research into context representation and awareness. Since there did not appear to be anything available, we developed a Jabber context browser that displays nodes and items in a Jabber pubsub system. The browser uses Echomine Muse (`open.echomine.org/confluence/display/MUSE`), a Java-based Jabber library. The browser is a Jabber client to the server, and the library handles the client-server connection and the incoming and outgoing XML communication at a relatively high level. We, as developers, only need to focus on how to react to each incoming XML stanza, and what should be in an outgoing XML stanza, instead of parsing and composing them. The browser also understands additional semantics we implemented to build a simple context service on top of pubsub [14]. For each abstract node in the context service, there are two corresponding pubsub nodes. One stores the payload of an item, while the other stores meta-information about the node, such as its MIME type and other attributes added by an application. See Fig. 6.

Health-care scenarios. We participated in project whose goal was to investigate how intelligent systems and ubicomp technologies could contribute to both research into cognitive impairments like Alzheimer's disease in the elderly, and to the care and comfort of cognitively impaired elders. Our role was to develop and deploy smart-space infrastructure in support of other project activities. For demonstration purposes, we



Jabber Server jabber.sl PubSub Server pubsub.j Username x10agen Password ●●●●●● Resource home Port 5222

```
pubsub
  Device
  WebCam
    motion (plain node)
    snapshot (plain node)
    snapshot (typed node)
  lamp
    1 (plain node)
  princely
```

Field	Value
Persist items to storage	1
Deliver payloads with event n...	1
The type of node (collection o...	leaf

ID	Item Attributes
2007-02-22T16:06:13.343-0...	`<sensor xmlns="http://jabber.org/protocol/pubsub`
2007-02-22T16:06:13.340-0...	`<sensor xmlns="http://jabber.org/protocol/pubsub`
2007-02-22T16:06:10.203-0...	`<sensor xmlns="http://jabber.org/protocol/pubsub`
2007-02-22T16:06:07.745-0...	`<sensor xmlns="http://jabber.org/protocol/pubsub`
2007-02-22T16:06:07.740-0...	`<sensor xmlns="http://jabber.org/protocol/pubsub`
2007-02-02T14:10:08.494-0...	`<sensor xmlns="http://jabber.org/protocol/pubsub`

```
Connecting to Jabber server...
Connected to Jabber server.
Discovering nodes from Pubsub service...
PubSub nodes retrieved.
```

Figure 6: Context browser

were able to prototype an elder-care scenario involving a "smart bed," webcams, X10 sensors, and a telephone system.

In the scenario, a hospital bed is instrumented with flexible ShapeTapes (`www.measurand.com/products/ShapeTape.html`) which report their shapes as they are deformed by a patient's weight and motion. The ShapeTape data is analyzed to publish Jabber events such as "bed empty" and "bed occupied." X10 motion sensors detect movement in a particular area near the bed, and publish motion events. We have have a webcam in place in order to take snapshots.

The ShapeTape analyzer, the webcam, and the X10 sensors are all Jabber clients of the same server, which also offers a pubsub service. The webcam subscribes to bed and motion events, and takes a snapshot when one of them occurs, publishing the image back to the context store. Health-care professionals are assumed to use Jabber clients and subscribe to the snapshots, as an aid to deciding whether to intervene.

Another scenario is that when an older adult turns on appliances such as a stove, he/she might forget to turn them off. So, when a potentially dangerous appliance is turned on or off, a phone call is made to inform a care-giver or relative. In this scenario, we use X10 modules and the PBX system. An X10 module connects to an appliance and appears to be a Jabber client. It publishes "ON" and "OFF" events to the pubsub server. A Jabber client application subscribes to these events and activates phone calls accordingly.

A smart walker We augmented a walker (with four wheels, handbrakes, and a seat) by attaching different Phidget sensors (`www.phidgets.com`) to determine the state of the walker, analyze patterns of how it is used, and help users avoid dangerous situations. We attach slidometers to both brakes to detect when they are applied and how far they are applied. We attach rotation sensors to two wheels to detect changes in direction of the wheels.

This work is currently intended only as a proof-of-concept sufficient to inspire a larger, ongoing, multi-disciplinary smart-walker project. We have a Jabber client that receives Phidget-sensor events from the walker and then publishes them to a pubsub server. Another Jabber client subscribes to these events and analyzes them. Currently, we simply translate raw values of Phidget sensors into messages intended for people, and send them to interested parties, via Jabber chat, email, or audio.

In summary, we have clear evidence that Jabber serves us well as a platform on which to build ubicomp systems with only modest investments in software development. We contend that it can also serve as the basis for the ecology on which to build second-generation ubicomp systems, and conduct further research, as described in the next section.

6 Research Activities

The smart walker demonstration described above is not research in and of itself; rather it shows that we can ignore the mechanics of provisioning and exploiting the ubicomp infrastructure as we concentrate on research issues in mechanical design (semi-autonomous braking systems?), patient care (coaching patients with dementia?, intervening when patient agitation is detected, before negative responsive behaviour occurs?), and physiotherapy (how much mobility has a patient demonstrated today and in the recent past?).

Less speculatively, we have also begun research into second-generation ubicomp issues. El-Sayed [8, 7] shows how one might improve on Jabber's single-system service-discovery protocol [XEP-0030] to provide a context-aware service-discovery architecture that exploits meaningful contextual information, either static or dynamic, to provide users with the most suitable and relevant services. The architecture relies on a shared, ontology-based, semantic representation of services and context to enable knowledge sharing, capability-based search, autonomous reasoning, and semantic matchmaking. Services can be invoked either via SOAP over XMPP [XEP-0072] or Jabber ad-hoc commands [XEP-0050].

Ubicomp research has long held that the systems we build should disappear into the background, and not be intrusive. We are investigating how to incorporate a technical notion of intrusiveness into a ubicomp system, based on simple numeric attributes of messages and entities receiving them. In our model, messages are assigned an importance between 0 and 1, and receivers publish their general willingness to receive messages, also as a number between 0 and 1. The model is then refined to modify the importance and willingness based on context, and, when importance trumps willingness, to deliver the message non-intrusively, also based on context. We use Jabber to build prototypes of message-processing algorithms for non-intrusive messaging.

We believe that the most promising way to standardize the meaning of "context" is to pursue an ontology-based approach using the current semantic-web framework, OWL [18], and related standards and tools (such as SOUPA and FOAF). These provide good facilities for understanding and reasoning about context, but not for timely dissemination of changes in context, as is required by many ubicomp applications. We have begun to look at how the ontological approach can be combined with pubsub semantics for communication [12], and have implemented a prototype based on Jabber pubsub.

7 Conclusion

Returning to Fig. 1, if Weiser's vision of ubiquitous computing is to become a reality, we need to foster the development of an ecology that allows many players to interact flexibly, with a shared understanding of common concepts, shared protocols, and a plethora of devices, people, smart spaces, and applications. Jabber allows us to do that.

Sensors can be wrapped easily to communicate with other entities that are interested, whether through a context engine, simple pubsub, or exchange of text messages. Actuators can be trivially wrapped to accept text commands, or more carefully incorporated to be discoverable and export an ad-hoc command interface, a Jabber RPC interface, or a SOAP over XMPP interface. The sensors, actuators, components, applications, and users can be discovered with Jabber's simple service-discovery protocol. Data can be stored, published, and subscribed to easily, with or without a more heavyweight context engine. Jabber's user and security models are "good enough" to use, although more research is still required to understand what "the best" approach is in general.

Jabber provides a rich set of communication paradigms, a distributed architecture of multiple servers, and clients, libraries, components, and servers for many, many platforms. The client software already comes with a familiar user interface that may be sufficient in many cases, although unusual interfaces could also be developed, supported by all the rest of the Jabber infrastructure. Simple pubsub can provide relatively unstructured context information, and serve as the basis for advanced research and development of more capable and intelligent context services. And we can write applications on top of a standardized, well-supported layer of middleware and protocols.

Finally, we can reach a point where we can exchange technical artifacts for rapid prototyping and even production developments of ubicomp systems. We can move forward to focus on research, while sharing our efforts with others, rather than reinventing the wheel. We need to start co-operating to show that our goals need not remain pipe dreams.

Acknowledgments Thanks to Herman Li and Abdur-Rahman El-Sayed for using and contributing to our Impress project.

References

[1] Extensible messaging and presence protocol (XMPP): Instant messaging and presence. URL `http://www.xmpp.org/specs/rfc3921.html`. RFC 3921, Peter Saint-Andre (Ed.).

[2] Extensible messaging and presence protocol (XMPP): Core. URL `http://www.xmpp.org/specs/rfc3920.html`. RFC 3920, Peter Saint-Andre (Ed.).

[3] Dan Brickley and Libby Miller. FOAF vocabulary specification. URL `http://xmlns.com/foaf/0.1/`.

[4] Harry Chen, Tim Finin, and Anupam Joshi. An ontology for context-aware pervasive computing environments. *The Knowledge Engineering Review*, 18(03):197–207, 2004. URL `http://journals.cambridge.org/action/displayFulltext?type=1\&fid=220542\&jid=KER\&volumeId=18\&issueId=03\&aid=220541`.

[5] Harry Chen, Filip Perich, Tim Finin, and Anupam Joshi. Soupa: Standard ontology for ubiquitous and pervasive applications. In *Proceedings of the First Annual International Conference on Mobile and Ubiquitous Systems: Networking and Services (MobiQuitous'04)*, 2004. URL `http://ieeexplore.ieee.org/iel5/9259/29413/01331732.pdf?arnumber=1331732`.

[6] Anind K. Dey. Understanding and using context. *Personal and Ubiquitous Computing Journal*, 5(1):4–7, 2001.

[7] Abdur-Rahman El-Sayed. Semantic-based context-aware service discovery in pervasive-computing environments. M. Math. thesis, David R. Cheriton School of Computer Science, University of Waterloo, Waterloo, Ontario, Canada. N2L 3G1, December 2006. URL `http://uwspace.uwaterloo.ca/handle/10012/2638`.

[8] Abdur-Rahman El-Sayed and James P. Black. Semantic-based context-aware service discovery in pervasive-computing environments. In *Proc. First IEEE Int. Workshop on Services Integration in Pervasive Environments*, pages 9–14, Lyon, France, June 29 2006. URL `http://www.cs.uwaterloo.ca/~aaelsaye/paper.pdf`. Held in conjunction with IEEE Int. Conf. on Pervasive Services.

[9] David Garlan, Daniel P. Siewiorek, Asim Smailagic, and Peter Steenkiste. Project Aura: Toward distraction-free pervasive computing. *IEEE Pervasive Computing*, 1(3):22–31, April–June 2002. URL `http://ieeexplore.ieee.org/iel5/7756/21806/01012334.pdf`.

[10] B. Johanson, A. Fox, and T. Winograd. The interactive workspaces project: experiences with ubiquitous computing rooms. *IEEE Pervasive Computing*, 1(2): 67–74, April–June 2002. URL `http://ieeexplore.ieee.org/iel5/7756/21806/01012339.pdf`.

[11] Glenn Judd and Peter Steenkiste. Providing contextual information to pervasive computing applications. In *Proceedings of the First IEEE International Conference on Pervasive Computing and Communications, 2003. (PerCom 2003)*, pages 133–142, March 2003. URL `http://ieeexplore.ieee.org/iel5/8487/26747/01192735.pdf`.

[12] Omar Zia Khan. Incremental deployment of context-aware applications. M. Math. thesis, David R. Cheriton School of Computer Science, University of Waterloo, Waterloo, Ontario, Canada. N2L 3G1, December 2005.

[13] Tim Kindberg, John Barton, Jeff Morgan, Gene Becker, Debbie Caswell, Philippe Debaty, Gita Gopal, Marcos Frid, Venky Krishnan, Howard Morris, John Schettino, Bill Serra, and Mirjana Spasojevic. People, places, things: web presence for the real world. *Mobile Networks and Applications*, 7(5):365–376, 2002. ISSN 1383-469X. doi: http://dx.doi.org/10.1023/A:1016591616731. URL `http://dx.doi.org/10.1023/A:1016591616731`.

[14] Herman Li. The Impress context store: A coordination framework for context-aware systems. M. Math. thesis, David R. Cheriton School of Computer Science, University of Waterloo, Waterloo, Ontario, Canada. N2L 3G1, April 2006. URL `http://etd.uwaterloo.ca/etd/hyh2li2006.pdf`.

[15] Anand Ranganathan and Scott McFaddin. Using workflows to coordinate web services in pervasive computing environments. In *Proc. IEEE Int. Conf. on Web Services*, pages 288–295, San Diego, California, July 6–9 2004. URL `http://ieeexplore.ieee.org/iel5/9185/29136/01314750.pdf`.

[16] Manuel Román, Christopher Hess, Renato Cerqueira, Anand Ranganathan, Roy H. Campbell, and Klara Nahrstedt. A middleware infrastructure for active spaces. *IEEE Pervasive Computing*, 1(4):74–83, October–December 2002. URL `http://ieeexplore.ieee.org/iel5/7756/25949/01158281.pdf`.

[17] João Pedro Sousa and David Garlan. Aura: an architectural framework for user mobility in ubiquitous computing environments. In Jan Bosch, Morven Gentleman, Christine Hofmeister, and Juha Kuusela, editors, *Software Architecture: System Design, Development, and Maintenance (Proceedings of the 3rd Working IEEE/IFIP Conference on Software Architecture)*, pages 29–43. Kluwer Academic Publishers, August 25–31 2002. URL `http://www.cs.cmu.edu/afs/cs/project/able/ftp/wicsa3-aura/wicsa.pdf`.

[18] W3C Consortium. Web ontology language (OWL). URL `http://www.w3.org/2004/OWL/`.

[19] Mark Weiser. The computer for the 21st century. *Scientific American*, 265 (3):94–104, September 1991. URL `http://www.ubiq.com/hypertext/weiser/SciAmDraft3.html`. Reprinted in *IEEE Pervasive Computing*, vol. 1, no. 1, Jan–Mar 2002, pp19-25. No electronic copy available for copyright reasons.