The Continuum Architecture: Towards Enabling Chaotic Ubiquitous Computing

by

Octavian Andrei Drăgoi

A thesis
presented to the University of Waterloo
in fulfilment of the
thesis requirement for the degree of
Doctor of Philosophy
in
Computer Science

Waterloo, Ontario, Canada, 2004

Author's declaration for electronic submission of a thesis

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

Interactions in the style of the ubiquitous computing paradigm are possible today, but only in handcrafted environments within one administrative and technological realm. This thesis describes an architecture (called Continuum), a design that realises the architecture, and a proof-of-concept implementation that brings ubiquitous computing to chaotic environments. Essentially, Continuum enables an ecology at the edge of the network, between users, competing service providers from overlapping administrative domains, competing internet service providers, content providers, and software developers that want to add value to the user experience.

Continuum makes the ubiquitous computing functionality orthogonal to other application logic. Existing web applications are augmented for ubiquitous computing with functionality that is dynamically compiled and injected by a middleware proxy into the web pages requested by a web browser at the user's mobile device. This enables adaptability to environment variability, manageability without user involvement, and expansibility without changes to the mobile.

The middleware manipulates self-contained software units with precise functionality (called *frames*), which help the user interact with contextual services in conjunction with the data to which they are attached. The middleware and frame design explicitly incorporates the possibility of discrepancies between the assumptions of ubiquitous-computing software developers and field realities: multiple administrative domains, unavailable service, unavailable software, and missing contextual information.

A framework for discovery and authorisation addresses the chaos inherent to the paradigm through the notion of *role assertions* acquired dynamically by the user. Each assertion represents service access credentials and contains bootstrapping points for service discovery on behalf of the holding user.

A proof-of-concept prototype validates the design, and implements several frames that demonstrate general functionality, including driving discovery queries over multiple service discovery protocols and making equivalences between service types, across discovery protocols.

Acknowledgments

I would like to thank my adviser, Dr. Jay Black for his suggestions, excellent advice, patience, and for securing financial support. I also wish to thank my committee members, Dr. Johnny Wang, Dr. Ajit Singh, Dr. Raouf Boutaba, and Dr. Roy Campbell, for their time and for advice in focusing the scope of this research and improving its presentation. Faculty and students in the Distributed Systems and Networks Group at the University of Waterloo provided a good work atmosphere and stimulating exchanges of ideas. Sébastien Brat helped with coding portions of the Continuum prototype, while working as a student researcher for Dr. Black. Completing this work would not have been possible without the continuing support of my parents, the support of valuable friends, and the excellent education I have received from several dedicated professors at "Politehnica" University Bucharest and teachers at Colegiul Național de Informatică "Tudor Vianu" and Şcoala Generală 104, both in Bucharest.

Table of contents

List of figures

List of tables

Chapter 1

Introduction

Ubiquitous services and devices are increasingly woven into all our activities, and our dependence on them will grow. This raises opportunities for middleware infrastructure to structure and facilitate interactions among a user's personal mobile devices, surrounding devices, and other local services.

The ubiquitous computing paradigm was introduces by Weiser's vision [132] of computing devices embedded in the physical environment, integrated with each other, and running applications that "follow" the user, being readily accessible from anywhere. The Xerox Parc project started early experiments on how to design suitable systems and devices [128]. Realizing this vision draws from many research areas including: location technologies and user tracking, wireless network-access technologies, innovative wireless devices, service discovery (including direct point-to-point discovery), sensing, capturing and expressing the context, software that adapts to mobile environment changes, software designed expressly to support ubiquitous computing, and domain-specific applications.

Ubiquitous computing users are essentially immersed in a space rich in electronic devices. Ubiquitous computing is not simply about computations on the devices, mobile devices, and sensors. It is about the user's access to her data and about what users can with neighbouring devices, in conjunction with that data. Often, the electronic surroundings will be unfamiliar: a different office, a different city, even a foreign country. The user will have to discover what is available in the current location, what credentials she needs, and where to get these credentials. These scenarios are "chaotic" in two senses. First, the user does not know how to access local devices, he might not have the needed software, and he might not have even noticed their existence. Second, the electronic surroundings do not all belong to one administrative domain and there is no single central authority. Among ubiquitous-computing scenarios, these chaotic

ones most need middleware mediation, a specific software framework, and specific mechanisms to help the user make sense of and use the surrounding electronic environment.

Technically, ubiquitous computing interactions are possible today, but not in chaotic environments. Here are several sample scenarios that each can be implemented currently in test-bed environments where the developer has control over the software on all the entities involved, but that cannot work *in vivo*, where the developer does not have such control.

- A guest speaker is to gives a talk in a small community center, not particularly well equipped. Through her wireless PDA (Personal Digital Assistant), she interacts effortlessly with the on-site internet-connected data projector. The evening before, in a local hotel room, from her PDA, she has remotely reviewed an outline of her presentation. She has also chosen to generate printouts, an option presented automatically by software. The printouts were ordered from a copy store near the hotel. In a university, a guest researcher uses his cell phone to acquire credentials and information on where to find local services. This information is acquired from the PDA of the host professor, while the two meet over coffee in the lounge. Subsequently, from his own tablet PC, the guest faxes a technical report using the departmental fax. The charge goes on the host's research account.
- In a shopping mall, a customer's mobile phone obtains discount coupons that a special service hands out to individuals on the premises.
- In a hotel, a guest interacts with the cash dispenser in the lobby. The amount appears on the hotel bill.
- In a hospital, from her PDA, a doctor orders X-rays from the patient's family doctor, and displays them on the large screen in her office.

All these examples imply that all the software required is already installed on the user mobiles and everywhere else needed, that it locates all devices, and that it knows their low-level and high-level semantics. For instance, the presentation example requires that the software and the user determine the

projector is available, the operations it can support, and how to actually execute the operations. One might argue that, if the software functionality to redirect the presentation is not available, the user can simply install a small software unit to do just that. However, in other settings, she might need to initiate other simple operations. This set is essentially unbounded set. A few examples include changing the picture displayed on the electronic poster in her living room, playing a music file she owns on the sound system of a friend, dialling a phone number contained in the web page she is viewing, and inserting the picture just taken by a co-worker with his digital camera into the document she is editing are just few examples. It is unlikely the guest speaker can herself install, manage, and upgrade all these small pieces of software, especially when many of them might be applicable in just a few locations that the user will never visit twice.

Today's ubiquitous-computing research largely ignores the chaos inherent to the paradigm. Projects such as Aura [116], EasyLiving [10], iRoom [53], and Gaia [104] focus on specially-designed smart rooms with a particular purpose and application domain (e.g., meetings, presentations). An example of a current alternative to smart spaces is the PrintMe™ system [93], a commercial system that allows guests of subscribing hotels to print from a mobile device to the hotel printers or to an affiliated printing-center located nearby, with, or without in-room delivery.

There are several drawbacks to the current state of the art. First, a specially equipped, homogenous environment (the smart space, the special hardware for PrintMe) is required. This is bound to prevent interactions with services outside the closed scope of the solution and in other smart spaces. Handcrafting a system based on simplifying assumptions about the discovery protocols in use, the hierarchy of services types in the system, the kind of activities the user will perform in the setting, and the pattern of interaction results in systems that, even if they are highly decoupled by structure, are monolithic by purpose.

The smart-space approach ignores the reality of multiple and not necessarily cooperating administrative domains, hence interactions as the examples above can be supported only in one realm. However, different local devices can belong to several administrative domains that overlap arbitrarily, as

suggested in Figure 1.1. Some devices might be in an administrative domain by themselves. For instance, a department outfits a presentation room with devices, while the university outfits the floor. While in the room, the user might also need to access devices elsewhere on the floor, bypassing the departmental administrative domain.

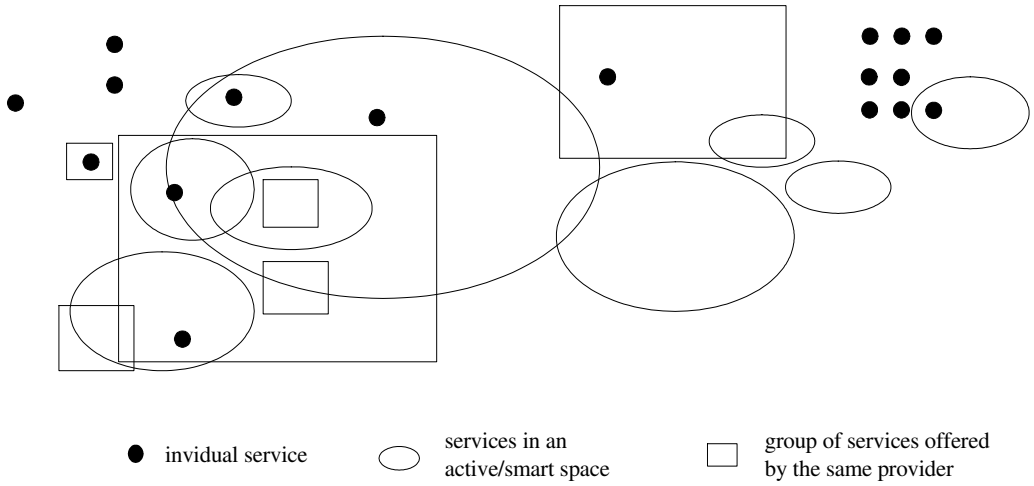

Figure 1.1. Multiple, non-cooperating administrative domains

Another characteristic of the current smart-space based approaches is that they tend to focus on one application domain. Deploying a solution in a particular setting, like a hospital, an office, a hotel, or a museum facilities assumptions on what user activities will take place there. However, this can result in systems that do not support activities outside that space. For instance, a hypothetical ubiquitous computing system deployed in a hospital would not support activities that need printers of the copy store nearby. One deployed in a university might not interoperate with one deployed in the mall down the street, and a visitor in the university would not be capable of accessing the services and devices made available by the city.

Given the lack of *a priori* relationships between peers, standardization alone is not enough to enable ubiquitous computing. Moreover, at this point the various smart space platforms are not compatible, in the sense that a mobile device will need new software installed for each of them, which is unacceptable if the

user moves frequently among spaces using different technologies. Privacy and confidentiality issues can make interoperation even harder to achieve.

The current research overlooks the need for a clear software abstraction as a key aspect of a scalable ubiquitous computing solution. Traditionally, a program is a rather large piece of software, distributed or not, which the user runs to carry out a certain function. In ubiquitous computing, this notion is not suitable outside closely-knit smart spaces. The ubiquitous-computing user will simultaneously carry on several activities and, often, neither their nature nor the relations among them can be foreseen by a developer at design time, so defining the different activities around the notion of computer programs is too restrictive. This becomes even more of an issue when the different programs are written by different software developers, since the individual agendas of users, SPs, ISPs, software developers, and content providers all need to be accommodated.

Often, it is taken for granted that the system would be able to deduce the user's intent and implement it proactively on her behalf. However, this ignores the fact that users change activities frequently and interleave tasks from different application domains in a random manner, out of reasons that are not technically quantifiable. For instance, a doctor, on his way back from a patient visit, might decide to pre-order a coffee at a nearby store (instead of the staff cafeteria), for an impromptu break and a walk in the park. Not only has the doctor changed his activity, but also the level of confidentiality has dropped, from the typically high requirements of an activity handling patient data to the less critical ones ordering a coffee.

The main drawback of current approaches to ubiquitous computing is that they do not enable an ecology among the players at the edge of the network. These players are the users, the service providers, the ISPs, the software developers, and content providers. The users can be known or unknown to the systems. The service providers can be hotels, cinemas, tour operators, print shops; small convenience stores, etc., and they have various sizes and level of technical expertise. The ISPs compete for users and have overlapping service areas. The software developers business is to augment the user experience with

the ubiquitous computing. We call enabling such an ecology among these players that do not necessarily cooperate with each other *chaotic ubiquitous computing*.

The *service providers* are administrative domains (e.g. companies) that deploy services, manage and maintain them, and benefit from their deployment. The benefit can be through direct fees collected from users (per-access or flat fees) or by ensuring that users (customers) return to a certain physical space (e.g., an airport lounge, a book shop) due to the richness of services in that space.

A *service* is a piece of functionality that has an impact on or depends upon the state of the physical world surrounding the user (e.g., a printer issues a printout, a scanner scans a document, a display shows something to the user). A service can have arbitrary semantics. It is offered by a physical device (physical service) close to the user or by networked computers (software service) with particular access and authorisation policies (e.g., requiring the user to be co-located, to belong to a realm, or to have some trait, such as being a guest or being a participant in a discussion, party, or seminar). The characteristics of a service are defined in detail in Chapter 3. Here we note that services can be as varied as a wall-mounted display, a parking meter, a printer, a cash dispenser, a digital camera, a PDA belonging to another user, or even specific software services within a physical space, such as a patient database in a hospital. While accessible over the internet, they often might be physical devices close to the mobile user and have peculiar access and authorisation policies.

1.1. Motivation and thesis statement

In the light of the drawbacks enumerated above, one can identify several problems that motivate this thesis. The user needs assistance to access services offered by competing service providers, across authority domains that overlap, in spite of having no single point of authority. Software needs to be structured so that it can evolve to keep up with the new types of contextual services that a mobile might encounter. Likely, services will be advertised over one of several service discovery protocols, each with its own growing set of service types. Software also needs to be written to cope with diverse and evolving environments, from very rich in contextual services to lacking many such services. Managing the

deployments and updates of such software is thus a problem. The ubiquitous-computing paradigm should be made available to users of mobile devices without changing these devices, the basic software on those devices, or the server-side software already in place (e.g., web applications).

To address these problems, this thesis proposes the Continuum architecture to bring ubiquitous computing to chaotic environments by enabling access across administrative domains without imposing a hierarchy or requiring agreements among them.

In a design based on this architecture, ubiquitous-computing functionality is made orthogonal to web-based applications, and is dynamically injected by a middleware proxy, requiring no changes to clients, and allowing for easier prototyping and deployment of support for additional ubiquitous-computing activities. The scope of the ubiquitous-computing functionality we consider is that of user-driven, short-term, ad-hoc interactions with contextual (ubiquitous) services (e.g., printing, giving a presentation, acquiring data). The interactions are not planned in advance, nor are the services known a priori. Conceptually, the data accessed by the web application or by the mobile user is put into its "physical" context.

1.2. The Continuum architecture, design, and prototype

Our Continuum architecture adopts a data-centric approach. Essentially, it advocates shifting away from the model of a meta-computer (e.g., a physical room) with a meta operating system, towards a model in which middleware attaches ubiquitous-computing functionality to the data that software or the user accesses, independent of the reasons for accessing the data.

Conceptually, the user experience with Continuum is as follows. While interacting with web applications or simply browsing the web, she can count on select data rendered in the browser window to be put in relation to the "physical" context of the mobile user automatically. Essentially this means that she can click through or otherwise access a toolbox of dynamic tools that can be applied on that data.

Selecting particular tools and associating them with data that will be rendered at the browser is called *publication*, a term more completely defined in Chapter 3. Different tools can potentially come from different software providers.

Each of these tools has precise functionality that helps the user interact with one or more contextual services, in conjunction with the data to which the tool is attached. For example, one tool to be applied to a picture viewed on the PDA could display it on a projector, as a user-friendly way to interact with this contextual service. A projector is a service with simple high-level semantics, but it is accessed over some arbitrary low-level protocol and understands only a small and specific set of data formats.

The self-contained software unit behind each tool is called *frame* and is defined in detail in Chapter 4. As we will see, frames are often implemented as an applet-servlet pair, first running within the browser, being a combination of HTML and JavaScript code, and the second running at the middleware, being written in Java.

Unlike other ubiquitous-computing systems, Continuum avoids guessing (and likely guessing wrong) what the user wants to do, and proactively doing it behind the scenes (and hence confusing and frustrating the user). Instead, it has the more realistic objective of unobtrusively presenting to the user the means to access and exploit her current electronic environment, leaving it to her to decide how to use these means.

Continuum involves the user in helping the middleware access information to better do its function. The middleware provides users with frames to dynamically acquire *role assertions* (defined in Section 5.1) issued by other users. The role assertions capitalise on the results in Role Based Access Controls (RBAC) research. As we shall see, they contain both information about the local electronic topology and credentials asserting that the user has a certain role. A contextual service accessed through the mediation of a frame can then authorise the access based on such credentials.

Conceptually, the Continuum architecture has four parts:

1. the middleware that dynamically publishes frames (tools), essentially injecting ubiquitous computing functionality;

2. a software-development framework centred on the notion of a frame;

3. a service discovery and authorisation service framework; and

4. a framework for dynamic acquisition and dissemination of *role assertions*.

Several design requirements were taken into account when realising the Continuum architecture.

1. There should be no need for special-purpose code to be preinstalled on user devices, and any code downloads during web browsing should be minimised and automated. This should require no active administration from the user and no changes to the client devices.

2. The solution should be easy to deploy incrementally, it should capitalise on existing legacy services, and it should allow graceful degradation of functionality when less infrastructural support is available. (A user should be assisted both in the small corner shop and in the fully equipped office.)

3. The design should require only local deployments, without any large-scale changes to the internet. The infrastructure to be deployed by service providers should be minimised and service offerings from multiple, competing, and perhaps small service providers should be encouraged.

4. When and if co-location of users, service providers, and ISPs occurs, it should be exploited.

5. The design should enable the creation of markets where users, service providers, ISPs, and software developers all realize value and obtain revenue.

6. The resulting system should be intuitive and non-intrusive for the user to use.

The design attempts to leverage current standards and technologies (HTTP, HTML, scripting technologies) and the current base of deployed hardware and software (software services, devices, handcrafted smart spaces, personal wireless devices). Consequently, the ubiquitous computing functionality is made orthogonal to web applications, and the data to which tools (frames) are associated is web pages and web objects. The middleware functions as an HTTP proxy that alters the HTML entities in transit to inject the additional functionality.

Several assumptions on the user model and personal mobile device capabilities enable the middleware to perform its function. At the conceptual architecture level, it is assumed that ubiquitous computing functionality can extend client-server applications. This means, in particular, that data accesses from the software on the mobile are exposed and that additional UI real estate can be obtained. It is also assumed that there are means to make the user aware that this functionality is additional and orthogonal to the initial functionality

At the design level, specialising the assumptions, it is assumed the user is interacting, through a standard browser, with a web application that does not use SSL. The browser it is assumed to supports HTML and JavaScript, as a standard mark-up language and a standard scripting language. The user interface for the *frames* that implement the ubiquitous computing functionality is displayed on demand in the browser window. The proof-of-concept prototype shows that these assumptions are feasible.

1.3. Thesis contributions

Unlike previous research, Continuum off-loads from the mobile the responsibility of having to manage the software it needs to cope with the chaotic environment. The user can access services in multiple domains without the need for the latter to organise themselves in a hierarchy or in super spaces. For an individual user, the following items implicitly define to what and how the middleware can assist access: bootstrapping points for discovery found in role assertions, the code base of tools available at the middleware, the time allocated to service discovery, and the publication heuristics and contextual information to which they have access.

The main contributions of this thesis are as follows:

1. Continuum makes the ubiquitous computing functionality orthogonal to other application logic. Existing web applications are augmented for ubiquitous computing by functionality that is dynamically compiled and injected by a middleware proxy in the web pages requested by a web browser at the user's mobile device. The approach ensures adaptability to environment variability, manageability without user involvement, and expansibility without changes to the mobile.

2. The frame software units are introduced, in a design that explicitly includes the possibility of discrepancies between the assumptions of ubiquitous-computing software developers and field realities: multiple administrative domains, unavailable service, unavailable software, missing contextual information, etc. The design ensures that, by writing a frame or a suite of frames, software developers can make independent contributions to the user's ubiquitous-computing experience, without interfering with the functionality offered by other parties.

3. The framework for discovery and authorisation addresses the chaos inherent to the paradigm through the notion of role assertions acquired dynamically by the mobile (the user).

4. The proof-of-concept prototype implements several frames that demonstrate general functionality, including driving discovery queries over multiple service discovery protocols and making equivalences between service types, across discovery protocols.

1.4. Organisation of the thesis

The dissertation is structured in eight chapters. The presentation and evaluation of Continuum generally follows three levels of abstraction: a conceptual architecture level (mostly in Chapter 3, and partly in Chapters 4 and 7), a detailed design level (Chapters 4, 5, and 7), and a prototype level (mostly in Chapter 6, and partly in Chapter 7).

Chapter 2 identifies general related work from distributed, mobile, and ubiquitous computing on which Continuum builds. Specific aspects of Continuum are then contrasted with particular related work. Chapter 3 clarifies the notion of context, with its system and user facets. Starting from that, it introduces the main elements of the Continuum architecture, with its user metaphor and middleware-provided ubiquitous computing functionality, orthogonal to other functionality. Chapter 4 devises a software abstraction on which such ubiquitous computing functionality can be built. Chapter 5 continues the discussion of design elements needed to realise the Continuum architecture. Chapter 6 focuses on the details of the Continuum proof-of-concept prototype that partially implemented the design. A critical

analysis of Continuum follows in Chapter 7, while Chapter 8 concludes the thesis and highlights the new research areas it opens.

Chapter 2

Distributed computing, mobile computing, and ubiquitous computing

The evolution of software from stand-alone computers to distributed computing, to mobile computing, and then to ubiquitous computing has had two constant challenges: coping with an increasing degree of dynamism in the runtime environment, and deciding how much of this dynamism is to be passed on to the user. Moving from single machines to networking and distributed systems brought more freedom from colocation with the computing and data resources [58]. It also required software to be more flexible in locating and using them. Wireless computing added user and device mobility. Software had to become aware of location, user device characteristics, and energy consumption patterns, and to adapt to changes in the environment [111]. Ubiquitous computing proposed a computing paradigm where computers disappear into the physical environment [132]. In smart spaces, ubiquitous computing interactions are a reality. However, software for ubiquitous computing should also be designed to cope with the large variation in service deployment outside such spaces. The goal is to work smoothly where fewer services are available, while also taking advantage opportunistically of the areas with good service deployment. Satyanarayanan calls these uneven conditioning and localized scalability [111].

This chapter relates the Continuum architecture to other work in ubiquitous computing, and puts everything into the larger context of distributed and mobile computing. The first section of this chapter looks at the evolution of distributed computing from trying to hide distribution to exposing it to the developer and sometimes even to the user, resulting in the success of the web, of component-based programming, and of the notion of composable software services. The second section reviews research efforts to create adaptable software for mobile computing. As a constant of research in this area, adaptable applications explicitly expose remote data access. This confirms the approach taken in the Continuum

architecture to attach the ubiquitous computing functionality to the data accessed by web applications. We also review research in adaptation of HTTP content, and of streaming content, taking place either at the server or at proxies. Section 2.3 groups related ubiquitous-computing research efforts into two areas: those that concentrate on building meta-operating systems and software for smart spaces, and those that concentrate only on enabling ad-hoc access to services. The Continuum middleware falls in the middle. It enables user access to contextual services by making the ubiquitous-computing functionality orthogonal to other software functionality, and proposing the frame software abstraction, hence identifying the role of the system in selecting and publishing frames. Section 2.4 compares the Continuum frame abstraction with other abstractions.

The use of contextual information is essential to ubiquitous computing. Section 2.5 surveys the context elements considered in the literature, from electronic characteristics of devices and services to characteristics of the physical world, including location. The section finishes with how such information can be captured and monitored, also including an overview of service discovery protocols as mechanisms to discover context. The chapter ends with a review of the security and privacy issues in ubiquitous computing. In particular, Section 2.6.2 presents work related to the role-based service authorisation and discovery framework in Continuum.

2.1. Distributed computing

We begin with distributed computing. Pioneering research in this area attempted to hide the distribution of resources from both the user and the programmer by designing systems and operating systems that made the distributed system look and act like one computer. Eventually, it has been recognised that distribution is not always the best option. For better or more reliable results, the user might slightly change his routine. With little additional effort, the developer might improve performance substantially.

Hiding distribution is not unlike the smart-space approach in ubiquitous computing, where a physical space (usually a room) is essentially built to act and be programmed as one computer. Moving away from

this can have similar benefits to what exposing distribution had in distributed computing. Essentially, this thesis explores how to build a system that takes this observation to its core.

In distributed computing, attempts to expose distribution resulted in the client-server paradigm promoting decoupling between components by committing to an interface (e.g., CORBA) not to an implementation. Component-based programming carried the ideas further, resulting in applications that are collections of self-contained units (e.g., Java beans, DCOM objects). To reduce complexity, heterogeneity can be masked using virtual machines and write-once run-everywhere languages such as Java and C#. These approaches are also used in ubiquitous computing. However, due to the impossibility of foreseeing at design time the technological and administrative entities with which the user will interact, the complexity and the degree of decoupling are even higher in this paradigm.

The web paradigm adapted the client-server model to a data-centric and user-centric (thin-client-centric) view of the world. Building on its success, the notion of Web Services [136] emerged, as small and reusable components that can be accessed over the web. They standardise the format of the messages (XML) and how they are exchanged, not just the signature of the interface, moving the service contract from the interface to the message structure.

Web services are an example of services, including software services. Standardisation efforts like Open Pluggable Edge Services (OPES) [2] illustrate ways to deploy such services. OPES is a framework for invoking, authorizing, and deploying distributed application services such as logging, accounting, and content transcoding, adaptation, and assembling. As another example, the Ninja project [38] proposes a platform organised around the notion of a data path. Transcoding services fit this view well. Ninja facilitates the implementation of scalable, robust, composable, and interoperable internet services. The NEC Multimedia System [88] experiments with QoS-based service contracts as an abstraction to present service composition to the user in a friendly way. As the user changes abstract parameters, the composition of the service is renegotiated and/or subcontracted.

2.2. Adaptable software for mobile computing

Returning to computing paradigms, with advent of mobile computing, the focus shifted from how to distribute, compose, and make the various components interoperate, to how to cope with intermittent connectivity and how to adapt to the dynamism of a mobile environment. The approach to mobile-application adaptability in the literature can be classified in four models: data-centric adaptation, such as the MPEG video player for the Odyssey platform [86], protocol-centric adaptation through the use of protocol filters [141], application-centric adaptation such as the speech-recognition application for Odyssey [86], and hybrid adaptation (any custom combination of the above).

The amount of work needed to make existing applications aware of mobility seems to indicate that traditionally designed applications might not be the best choice when mobility is involved. Joseph *et al.* [54] suggests that mobile-awareness is a direct application of the end-to-end argument to the case of mobility adaptation, and the system should just provide mechanisms and policies for several general cases, complemented with an API for information transfer between layers.

To work around the inflexibility of traditional software units, the Prayer computing environment [3] organises adaptation around adaptation blocks, procedures similar to critical regions, executed within a QoS class, e.g., LAN, wireless LAN, SLIP, or no connection. For each class, the programmer specifies an action such as block, best-effort, abort, or rollback, for all the possible shifts of QoS-class.

Other projects have experimented with the design of software components that can be included in an adaptation feedback loop, e.g., Romulac [7], Quasar [48], Agilos [69]. Agilos attempts to build general-purpose middleware to control the adaptation of target applications, unrealistically modeled as linear functions–a limiting factor. Mobiware [13] introduces the notion of a utility function to characterise the adaptive response of a program.

As early as 1994, the WIT Project [129] proposed to clearly expose data units, and evolved into a wireless application design based on revealing network costs, providing hints of future references, using a local proxy, and adapting to connectivity variations [130]. Subsequently, Odyssey [85] centered the

adaptation on the notion of data fidelity–the degree to which a copy of data presented for use matches a reference copy. Type-dependent components maintain the level of fidelity for each data object, while the adaptation policies are supervised by a type-independent component and by applications. Other projects organise adaptation around the accesses to a central data store [27, 23] or accesses to server objects [22]. The Rover computing toolkit [54] clusters adaptation around any request-reply interactions, enabling transparent queuing, reordering, and filtering of the RPCs. Puppeteer [66] proposes application-specific, component-based adaptation. A proxy exploits the API exposed by client applications to modify or replace parts of the structured documents accessed.

Adaptation based on manipulation of HTTP messages is particularly relevant to Continuum. The HTTP transcoding that was first advocated by Brooks *et al.* [9] has currently matured and it is part of commercial products, like WebSphere [131]. Subsequently, many projects studied web-content transcoding techniques and heuristics [4, 15, 33, 34, 40, 73]. To avoid inadvertent interference with the usefulness or intended message of the content, it has been proposed that the content provider make additional information available by attaching adaptation policies [76] or by enriching the content with RDF [127] meta-data [84]. The current tendency on the web to migrate to XML multiplies the opportunities for transcoding and other adaptation of individual regions of a web document identified by the XML tags.

As an alternative to transcoding, the browser can negotiate the format of the requested document with the server. An HTTP request can include a quality coefficient for each characteristic accepted (language, encoding, MIME type) [44]. This helps the server return the version that best matches client capabilities. These capabilities can be formalized, for instance, as a Composite Capabilities and Preferences Profile (CC/PP) [124, 125, 126].

Adaptive actions can take place at the server, at the client, or at a proxy. The proxy can run at the edge node of the network, split between the edge node and the client [45], at some node in the network, or, in

the case of streaming media, at several nodes where filters are dispatched dynamically, on the path from server to client [13, 140].

The Transend [33] project defines the basic adaptive actions a transcoding proxy can take: transformation, aggregation, and customisation of data (the TACC model). Transformation means data transcoding, and aggregation means collating data from multiple web pages into one; customization means delivering information relevant to the specific user that made the request. Proxies can also prefetch data to improve latency for clients connecting over low-bandwidth links, or do they can do file hoarding to help clients cope with intermittent disconnection (e.g., the Coda file system [112]). Other adaptive actions at the proxy can include differential transfer, pipelining requests, and reducing header verbosity [45]. The Columbia proxy system [142] explores general-purpose filtering with dynamically installed, general-purpose proxy components for low-level (IP, ICMP, UDP, or TCP) and high-level (MPEG, SMTP, or HTTP) protocols. Similarly, low-level base-station proxies in the Comma project [71] provide TCP connection maintenance during disconnection intervals, TCP stream prioritization, filtering, and sequence renumbering [59].

2.3. Software abstractions for ubiquitous computing

We have discussed several of the aspects in distributed and mobile computing. Continuum considers the ubiquitous computing functionality orthogonal to other (web) application functionality. Consequently, the former can be dynamically associated with the data the latter manipulates. This falls under the responsibility of ubiquitous computing middleware. In this section, we survey how several research projects choose to structure the ubiquitous computing interaction, and what software abstractions, application definitions, and structuring techniques they employ.

Analysing the needs of ubiquitous-computing software to enable and cope with physical integration and spontaneous interactions, Kindberg *et al.* [61] concludes that such software should be data-oriented. Other papers, such as the position paper for the Portolano project, embrace a similar view [32]. The software must take into account that the set of users and hardware components is highly dynamic, and it

has to identify clearly the responsibilities of the user and of the system. In that respect, Continuum's data-centric architecture clearly identifies that the system should present to the user a set of artefacts to access the electronic services around her, while she can provide additional input to refine the choice of artefacts in this set, as discussed in the next chapter.

Many ubiquitous computing-research projects [10, 53, 103, 116] are geared towards designing meta-operating systems and software platforms for ubiquitous computing applications, and they assume interactions happen only in smart spaces: well-delimited specially designed physical spaces rich in networked devices, under the control of one administrative domain. Smart spaces that use different approaches do not interoperate.

In the EasyLiving project [10] the smart space is the user's home. Maintaining a variety of information, including geometric information, such as "the user is in the viewing area of a certain display and is facing it," the system aggregates simple I/O devices for particular purposes.

The iRoom project [53] implements a middleware operating system (iROS) for a meeting room with large displays, centralised around an Event Heap (a Linda-like pool of events that ties together the space) and a Data Heap that stores semi-permanent data. SmartPresenter is an example of software running on top of iROS.

With similar goals to iROS, the Gaia OS [103] manages the devices and the software components in a physical space augmented with networked devices. A Gaia application is defined as a collection of services that cooperate to achieve a common goal, and it depends heavily on the space for which it is written. An application has a model component, a presentation component, and a controller that drives "sensors" that provide input, all under the control of an external coordinator. Bridge components act as a presentation component for one application and a sensor-input component for another.

With the notion of longer-lived tasks like writing a paper or buying a home, the Aura project [116] moves away from the application (a computer program) as a first-class object at the level of the ubiquitous computing system. A task is represented by a description of the abstract characteristics of the services it

needs, and executes in a so-called environment—essentially the set of devices and programs accessible in the given smart space. When the user moves into another smart space, under a different administrative authority, he is followed by a so-called aura—his tasks, and the system strives to (re)negotiate task support based on the services available in the new environment. This implicitly assumes service characteristics can be abstracted in the general case, but such an abstraction is likely to reduce to the lowest common denominator. Moreover, Aura's approach ignores the fact that there can be major differences in what services are available in different smart spaces, that users might resent using suppliers with different user interfaces (e.g., switching from Emacs to Microsoft Word or from OpenOffice Writer to vi), and, more important, that the user's tasks change in subtle ways with the available services. For instance, the *EditingToPresentation* task might become *ListeningPresentation* as the user boards her car and starts driving, which prevents her from entering text or watching a screen, but not from thinking about the presentation structure. This means the task will now need speakers and a text-to-voice converter, not a text editor and a screen.

In the One.World project [39], the core abstraction is the notion of an environment, recursively defined as a bundle of software, data (a tuple store), and nested environments. Environments can migrate, which means all references to external services are re-bound to newly discovered instances. Service discovery is based on broadcast and multicast, which is unsuitable to find services physically close but not on one's network. To allow environment migration, all programs are written in the same language (Java) on top of the One.World environment, limiting the ability to leverage already-deployed facilities. (For more about the challenges of deploying ubiquitous computing systems, see Davies *et al.* [24].) The One.World applications are divided into logic and operations, sharing with Continuum the notion that the actions on content should be separated from the content. The logic includes computations that do not fail, such a creating a text message, while operations are interactions that might fail, such as sending a message to a recipient. Note that One.World uses the term operation differently from this thesis (our definition is in Chapter 3). Just like any other software, Continuum frames can be structured as envisioned in One.World.

The Adaptive Personal Mobile Communications project [57] adopts a more heavy-weight structure for the system. Each entity (user, device, resource) has an associated mobile agent that supposedly deals with user mobility. These agents react to changes in the type and availability of resources, based on the knowledge they acquire about external entities. The decision can be based both on the history of previous behaviour and on simulation of possible interactions. The knowledge about each entity is structured as so-called mobilet, and is expressed in XML. A mobilet can be *teleported*: for instance, when a person moves to a hotel room, the mobilets corresponding to home devices can be mapped onto the devices in the hotel room. The scalability of the approach is severely limited by the implicit need for the observing agent to know quite a lot about the observed entity.

We now move on to review research that concentrates primarily on enabling access to contextual services rather than offering a complete solution. The CoolTown project [60] associates devices, people, and objects with a web presence. Users see these URLs as pointers to information on how to access the corresponding entities. Personal devices can sense such URLs directly from the environment (e.g., reading barcodes, listening to infrared or RF beacons) and can also beam them around. It is up to the recipient to treat the URL as data or as the URL of a service. The starting point for a user in a new area is the notion of a place, loosely similar to a smart space in the sense that it has physical boundaries and clear semantics of use. A place manager (web server) serves the web presences for the entities in the place. This has the drawback that the owner of the place must statically maintain the set of entities it wants accessible to the users in its space, including links to other spaces, limiting access to other administrative domains. Nevertheless, the CoolTown mechanisms for physically discovering web presences can nicely complement Continuum's role-based framework for service discovery and access across administrative domains.

Several other projects concentrate on service access, and, like Continuum, they organise the systen around data access. Users of the Satchel system [62] browse their PC directory structure from any Satchel-enabled location—a specially fitted office environment where a local Satchel server has been

deployed. Through a context-sensitive UI, they can apply office-related operations to documents, e.g., print, fax, beam, fetch, or scan them. The device manipulates tokens representing documents. A token is a document URL, digitally signed and timestamped to prevent replay attacks. The requests to apply services are communicated to the Satchel server over GSM, or through infrared receivers glued to the surface of the target physical devices and wired directly to the Satchel server, which acts as a hub. In the first case, preconfigured built-in services are used. In the latter, the server knows from which infrared port the request has come, so it can infer which service is the target. This tightly-coupled view of the world, the need for specific hardware, and the need to initiate interactions from within dedicated software installed at the mobile differentiates Satchel from Continuum. The M-links project [114] proposes a "do and dig" model for navigating web content from mobile devices. The user can navigate a link (dig) or apply a service from a list (do) that, in contrast to Continuum, is static.

Sometimes the mobile needs to access devices that are not available. To compensate, in some cases, one can create virtual devices with a similar role. For instance, a PC without a sound card and a telephone can together form a composite device that plays movies with sound [89]. EasyLiving [10] has such goals. The Composite Device Computing Environment (CDCE) [89] also tries to create virtual composite devices. The user explicitly accesses the CDCE, which manages the composable devices and knows in advance the device mix in the environment. There are few instances where one can compose devices to achieve the functionality of another missing device, so, Continuum does not attempt to create composites. A frame might use several external services, but this is done to implement an operation, not to simulate another device.

In mobile and ubiquitous computing, often, the mobile personal devices are seen as universal interactors. Pham *et al.* identifies three possible modes of interaction between a mobile device and an external service: abdication, in which the mobile relinquishes control; cooperative, in which both controls on the mobile and input controls on the output device can be used to control the application; and

exclusive, where only the mobile is used for control. This taxonomy is useful in illustrating the choices the developers of Continuum frames have in implementing them.

This interactor metaphor has spawned research on how to move the primary user interface back and forth from the personal device to surrounding devices. A futuristic example is the steerable interface [90]. It can move on the surface of ordinary objects, where the user needs it. In the end, he might not even need a special-purpose device, which means that, for practical purposes, the component that drives such an interface acts as a soft version of the personal device.

To facilitate migration of interfaces, researchers have also looked at how to generate them dynamically. For instance, Hodes *et al.* [43] investigates a framework for generating user interfaces (UI) for remote services, for mixing dynamically-generated UI with service-provided UI, for generating UI for subsets of arbitrary functionality, and for remapping an existing UI to a UI rendered on a different device. The project uses a special Interface Specification Language that combines UI modeling with mapping software interfaces to UI. Having a standard user-interface description language as an alternative to software interface types reinforces the idea that a software building block for ubiquitous computing should interface directly with the user, one of the foundation ideas in the design of Continuum frames. Similar ideas to those in Hodes *et al.* [43] are employed by the ICrafter component [92] within the iRoom project [53]. This time, the resource-limited clients are freed from having to deal with interface generation. A middleware layer working with the centralised context memory of the space generates a UI that can include user-friendly references to devices, such as "in the middle of the room." The interfaces are rendered on the user's device, and are generated based on device capabilities with one of the supported toolkits: HTML, WML, VoiceXML, or Java Swing. If no handwritten UI is available, ICrafter generates a simple interface in HTML or in Java Swing. A "Room Control" window with a room map identifies the location of the available services. This centralised design is not conducive to using services outside the iRoom. The user can use drag-and-drop to move data between services or she can obtain a generated UI for each service (e.g., a window showing all the controls of a data-projector service).

While much of the research looks at how to apply services to the data the user fetches, the Appliance Data Services (ADS) [47] looks at the complementary problem of how to apply services to data generated by single-purpose user devices. The project shares with Continuum the notion of separating the code selection from the data on which the operations will be applied and from the device or devices that handle that data. In ADS, the basic unit that is manipulated is a triple of userid, command-tag, and data to which the command refers. These units are generated by user devices but are acted upon by an infrastructure built on top of the Ninja architecture [38].

Most projects deal with managing services and facilitating access to them. The Rome architecture [46] observes the importance of infrastructural facilities to use data automatically in the situations where it is contextually most useful. In an example from Huang *et al.* [46], an intelligent fridge makes a note when the user removes the last can of pop. Later, when the user drives by a grocery store, the on-board computer automatically notifies the user that he needs to buy more. Rome is centred on the notion of a trigger. The infrastructure decides when to fire a trigger based on the spatial and temporal context. It also decides to which user devices to deliver the notification. The functionality is orthogonal to the goals of Continuum. However, we do note that the Continuum home node (defined in Chapter 3) is ideal to act as a Home Base [46] for triggers.

The Pervasive Content Distribution system [68] experiments with reducing latency through pre-processing and pre-distributing content to edge servers close to the user. Supposedly, following access patterns, the system infers what transformations to apply and what data to pre-distribute. In fact, the functionality has been developed as a context service within the ContextSphere architecture [68] for context distribution. Cherniack *et al.* uses user profiles to facilitate "data recharging" [17].

The relation between prefetching and trust has been noted by Satyanarayanan [110]. Locations close to the user cannot always be trusted not to tamper with the prefetched data. Consequently, the mobile itself should hoard digests for the data items it might need in the future, to make sure the prefetched data

has not been altered. If privacy is an issue, the prefetched data might also be encrypted, so the mobile also has to hoard the keys to decrypt it [110].

The delivery of contextual notifications to users is a particular instance of data reaching users ubiquitously, just like users access data ubiquitously. To facilitate this, the Mobile People Architecture [74] has an additional Person Layer added to the traditional network stack, which, instead of the device, makes the user the end point of a communication. A trusted Personal Proxy knows the user's whereabouts and is responsible for dispatching communication to her. The relation between the user and the Mobile People Personal Proxy is similar to the relation between the user and the home node in Continuum.

2.4. Continuum frames vs. other abstractions

We have reviewed several of the ubiquitous computing projects and the issues on which they focus. In this section, we contrast the Continuum frames with other software abstractions in the literature. Chapter 4 further justifies the need for such an abstraction and presents the functionality a general frame includes.

In our view, long-term tasks, such as those that are first-class entities in the Aura project [116], can be arbitrarily complex and often they cannot be characterized statically. Therefore, frames define simpler operations, which are not long-lived, do not require re-binding, and can be characterised statically.

The recombinant computing project (Speakeasy) [28] uses the notion of task templates, less complex than the Aura task. A Speakeasy task template specifies the data connections among a set of services that accept and generate data objects. A task template can be predefined, or the user can assemble it manually. From a system perspective, Continuum frames are more active than Speakeasy templates. As shown in Chapter 4, they contain code that can interact with the Continuum server to locate suitable services, and code that decides if a frame should or should not be published. From a user perspective, a frame does not perform a task, but an operation on a data object, which in trun advances the user's task. To achieve the same results as certain Speakeasy tasks, conceivably, a user might have to interact in sequence with several frames.

The Speakeasy project promotes interoperability [29] by assuming services have minimal, "recombinant" interfaces for data transfer, metadata, and control. This semantically neutral interface is supposed to work around the need for the mobile device to know about more interfaces. While such standardisation is useful, it overlooks the fact that services have high-level semantics beyond their software interface. In Continuum, the frame has the responsibility to convey to the user the high-level semantics of the operation implemented by a frame, and indirectly, the high-level semantics of the external services the frame is using.

Be they Aura tasks, Speakeasy task templates, Continuum frames, or even full-fledged applications for particular smart spaces, all these software abstractions represent another level of indirection in the user's interactions with services. The Continuum frames represent "glue code." Another example of glue code is the bridge notion in Gaia. However, while bridges mediate between applications (the user is not involved), frames mediate between the user and a small set of services. Moreover, bridges are installed explicitly, while Continuum frames are published dynamically by the middleware.

Frames as tools on data are a concept similar to notions such as tools, task panels, smart tags, and context menus in office productivity suites. However, from a system perspective, there are several fundamental differences in requirements and in how Continuum frames are created, published, and executed. Frames use external services. These services are not under the control of the designer of the frame and they need to be discovered, because frames cannot use built-in references or count on a well-known registry on the machine. The publication of a Continuum frame is often determined by the services available at the current location not by user's activity, because this is not as well defined as in office productivity suites. Furthermore, unlike the other artefacts mentioned, the frames are published and execute in heterogeneous environments, with different administrative domains.

Compared with traditional software units, frames can have the same functionality as proxies, but, unlike proxies, they actively search for the external services they need and can decide for themselves if they should be published. Unlike drivers, frames interface directly with the user, not with other software.

Stand-alone executables have richer semantics than frames. Web Services [8] are on par with other software services used by frames, and so are DCOM objects and JavaBeans. The Continuum frames are different because they interface directly with the user. Web Services commit to a message structure, while DCOM objects and JavaBeans commit to a software interface.

UPnP [78] also has the idea that users could interface directly with the device through a web page. The hosting program (called a User Control Point) acts as a browser, while the accessed service (called a Controlled Device) acts as a web server that allows changes to the state of the service. When a Continuum frame is interacting with only one service, its frame UI has the same purpose as the UPnP service interface. This is also similar to the idea of mapping UI controls to certain functionality offered by services, as in the work of Hodes *et al.* [43]. However, frames differ in that they often operate on the data to which they are dynamically attached, and have additional functionality related to converting that data, finding services they need, and deciding on their publication.

Continuum frames can run split, partly at the browser and partly at the middleware server. The part at the browser represents a simplified case of mobile code in the sense that it is dynamically added to web pages in transit to the user. Compared to active-network capsules [121], Continuum frames have a broader scope, they are not disseminated in the network (only at the client and the Continuum server), they do not require changes in routers, and they affect only application-layer data objects. Compared to mobile agents [18], the frames are not proactive, autonomous, or mobile, other than transferring part of the frame to the client; they execute under the user control, and do not collaborate with each other. However, they do decide for themselves if they apply to a certain data object.

2.5. Capturing the context in ubiquitous computing

Continuum frames, and in general, all software for ubiquitous computing rely on context information. In Chapter 3, we discuss this complex notion for Continuum. In this section, we survey what the literature considers as context elements, followed by what mechanisms have been proposed to capture, aggregate, and deliver the context to the interested parties (including the user).

The parameters of the communication with the mobile have been often considered part of the context. They include anything from data rate (throughput), available bandwidth, delay (latency), message round-trip time, SNMP statistics [59], maximum idle time for a client-server software interface [22], handoff-dropping probability and signal strength [119], to the cost of communication (cost-per-byte [3, 22] or cost per transfer of content-unit).

Device characteristics such as display size and colour depth, and software characteristics like the browser model and its capabilities (mark-up and scripting languages supported) have also been considered part of the context. More sophisticated device characteristics like orientation, speed of movement, or acceleration can also be considered, as well as runtime characteristics, like the amount of volatile memory and free volatile memory, the size of permanent storage or of free permanent storage, CPU load, an estimate of the computational power of the processor, memory or disk access time, or energy-consumption [72].

2.5.1. Service discovery as a instance of collecting contextual information

The set of devices and services that surround the user, and their characteristics, are also context. The hard part is how to discover these services and how to decide which services are actually near the user or relevant in some other way. Many service discovery protocols have been standardised. Likely, different administrative domains will use different discovery protocols.

Some of these protocols have been designed specifically for zero or close-to-zero administration in so-called home networks (networks of devices and computers within the user's home or office). The Jini technology [120] is designed for Java. Universal Plug and Play (UPnP) [78] is designed for device-to-device connectivity and proximity networking. The client only has to know the URI (Uniform Resource Identifier) for the desired type of service, so UPnP does not have a service registry. With similar goals, Bluetooth [6] also specifies its own discovery protocol for proximity networking. Another category of protocols is designed to scale within an organisation without necessarily requiring the client to know the service directory URL in advance. The Service Location Protocol (SLP) [100] requires direct interaction

of the client with a certain service directory, unlike Salutation [105], which has service brokers (Salutation Managers) to take care of this functionality. There are also general-purpose directory-access protocols, such as Light Weight Directory Access Protocol (LDAP) [98] and the more complex and less used ISO X.500 [138], all more suitable for white-pages-like listings of services, with frequent reads and infrequent writes. Recently, in the context of Web Services [8], Universal Description, Discovery, and Integration (UDDI) [123] has gained popularity as a business registry for such services, logically centralised, but potentially distributed physically.

The service type is specified by the particular service-discovery protocol. This is not to be confused with the high-level semantics of the service, as services with the same high-level semantics might have different types (e.g., a Jini printer and an SLP printer). This is one of the problems solved with the introduction of Continuum frames.

Many discovery protocols go further and attempt to impose interoperability between services. Jini enforces the Java language and the programming model of accessing services through de-serialized Java objects that know how to interact with the actual device. UPnP maintains interoperability between clients and the server by standardising the transport protocol to HTTP, and assuming that each service will also expose a user-accessible HTML interface. The XML files that describe how to access a UPnP service can also contain scripts. Bluetooth specifies a small set of application profiles. Salutation offers transport independence by performing all operations through a Transport Manager. UDDI relies on the Web Services approach of standardising the format of the messages accepted by each service. Some protocols are not concerned with what their service directories store (e.g., LDAP, X.500) or assume each service type specifies the access protocol (e.g., SLP).

From a ubiquitous-computing perspective, the problem with having all these different discovery protocols is not only the different hierarchies of types and attributes, but also the variable degree of precision with which one can specify the services. In Jini, a service type is a “well-known” Java interface. More characteristics are specified in an array of serialized objects of class *net.Jini.lookup.entry.Entry*, on

which byte-level direct matching is done. UPnP describes service types as URNs (Uniform Resource Names) specifying a type and a subtype. Bluetooth defines application profiles to differentiate between service types, while SLP defines standardized service templates specifying attribute names to be used in queries. Salutation defines a specification for each service type, including attributes that characterise the service. In an LDAP directory, entries are organised hierarchically following geographical and organisational boundaries, and then by hierarchical object classes with associated attribute schemas. A similar approach is taken in X.500, to which, initially, LDAP was designed as a front end. UDDI groups Web Services around their owning business entity, rather than their type.

One of the hard problems in ubiquitous computing is how to bootstrap service discovery. This is partly addressed in this thesis through the role-assertion framework for service discovery and acquisition. Where to start service discovery has a direct impact on what services are considered contextual. Essentially, bootstrapping service discovery means finding URLs of local service directories that contain services relevant to the user. Tightly coupled environments like smart spaces rely on one well-known directory service or a centralized repository of service advertisements. For instance, in the iRoom project [53] a centralised context memory holds the physical topology and the description of the space and user devices (the methods offered by each service and their parameters, as well as other characteristics). Each service periodically beacons its description into an Event Heap accessible by all entities in the space.

Some service-discovery protocols also include mechanisms for the discovery of service directories, but they do not work well with multiple administrative domains. Usually some form of network broadcast or multicast (Jini [120]) is used for this purpose, and often also to find services directly (e.g., SLP [100], SSDP [37] in UPnP [79]). Unfortunately, “close” in networking terms does not always mean physically or logically close. The mobile device might be on a different subnet than the local services andservice directories. This is likely if they use different means to connect to the network. (To complicate things, two devices with IP addresses on the same subnet might not be physically close at all.) Integration [101] of SLP [100] with DHCP [96] assumes there is only one service directory useful for the specific subnet on

which the mobile device resides. The URL of that directory agent (a service directory in SLP) is obtained from the DHCP server, at the same time as the IP, gateway, and DNS information. When multiple service directories are involved, it is likely that each is administered by a different entity, so at login time, the DHCP server will have to choose one, or provide more than one URL. In an effort to accommodate all potential clients, this might degenerate into too many directory services being provided to the device at login. The mSLP [102] experimental extension of SLP proposes as a work-around a scope-based, fully connected peering between directory agents. This implies either that all directory agents are managed by the same entity, or that some sort of trust relationship exists between the directory agents (hence between service providers).

2.5.2. User's physical world and user's location

In addition to services and physical devices, other physical characteristics, like light level, noise level, vibration, temperature, pressure, etc., are sometimes considered part of the context. In particular, the user location as an element of the context is particularly difficult to work with, and subject to research in several projects. A solution to this problem is beyond the scope of this thesis.

There are many technologies to determine location, some tied to particular network access technologies, some requiring network assistance (e.g., GSM), others relying on the terminals (e.g., detecting location beacons) or on third parties (e.g., GPS). In recognition of this, the Location Interoperability Forum, now part of the Open Mobile Alliance [87], aims to define a common solution to how mobile devices access location information, independent of particular technologies.

The United States Federal Communications Commission (FCC), through the Wireless Communications and Public Safety Act of 1999 (911 Act), has regulated that in cellular networks (GSM, CDMA), the location of the handset must be reportable with an accuracy of 50m for 67% of calls and 150m for 95% of calls (in the case of handset-based solutions), and 100m for 67% of calls and 300m for 95% of calls (for network-based solutions). For short range cable-replacement technologies (Bluetooth [6]) and for WLANs (WiFi/802.11), the location is usually given by the fixed location of the access point,

so it is similar to their range, up to 10m for Bluetooth, and up to 46m indoors or 92m outdoors for WiFi. For conventional GPS (Global Positioning Systems), the accuracy can range from 10m in rural areas to up to 100m in the city.

The location of services and devices is also relevant. For fairly stationary services, other than as above, the location can be specified as a property in a service directory. For instance, the SLP [100] template for printers uses the *printer-location* attribute, while Jini [120] expresses location as an instance of the class *net.Jini.lookup.entry.Location*. Other protocols do not have the notion of location (e.g., Bluetooth or UPnP), equating "close" with network proximity, often considering that close entities answer broadcast queries first. Some protocols have the notion of a logical administrative domain, which sometimes can be assimilated with the location, or can be used to infer location (e.g., the SLP SCOPE, the business address in UDDI, or the geographic information in LDAP directories). The user-relative location of a service can also be useful, but most service discovery protocols do not support such relative queries. Li [70] attempts a work-around, with a geographically-oriented internet service-discovery architecture that augments current service directories with spatial information, through the deployment of a hierarchy of geometry managers. In this way, service-discovery queries can also specify an area of interest.

Often, the format in which location is expressed depends on how it is modeled: hierarchically (rooms are in buildings, which are in cities, etc.) or absolutely (X, Y, and potentially Z, latitude and longitude, etc.) In the general case, ubiquitous computing solutions will have to deal with different location formats and conversions (e.g., converting the latitude and longitude coordinates to/from room numbers). In the context of the Aura project, Jiang *et al.* [52] tries to combine the two models with a unified location identifier. The execution of these location operators relies on specific Aura services in the network, which are supposed to have complete information about the topology of the space. A similarly centralised approach is used in the EasyLiving project, where a special database keeps the topological representation of the physical smart space and answers location queries [10]. Likewise, CALAIS [81] has a service that can answer location queries and can issue location events. It tracks location with several location

technologies, and maintains pre-recorded spatial information on containers (e.g., rooms) and junctions (e.g., connecting containers, hallways).

The problems of how to obtain the location, how to represent location, and how to convert between its different representations are beyond the scope of this thesis. In order to ensure interoperability, ideally, there should be a standardised way to represent location. However, this is a difficult endeavour because accuracy varies widely from one technology to another.

2.5.3. Context acquisition and monitoring

Since dealing with contextual information is generally needed in mobile computing and ubiquitous computing, issues like monitoring the environment, collecting and aggregating higher-level knowledge about the environment, managing contextual information, and delivering it to interested recipients have been studied extensively in the literature. Dey [26] introduces the notions of Context Widgets, Aggregators, and Interpreters that prepare context for other software units. Kidston *et al.* experiments with Environment Monitors [59] to compile data about the environment. The Context Service [68] in the ContextSphere project is an extensible middleware that enables the collection and dissemination of context information. Solar [16] is a distributed infrastructure with a similar goal. It executes application-modifiable graphs of operators (processing modules) to collect and synthesize the context.

In smart spaces, the problem of collecting and managing context information is significantly simplified, at the expense of interoperability and of operation across multiple administrative domains. For example, Aura employs a Contextual Information Service [55], while iRoom [53] has the event pool. An alternative approach to collecting context is taken by the CAPEUS project [108]. There, the application describes the contextual constraints and data that are required for the execution of a certain type of service, it packs them into Context-Aware Packets, and it releases them into the network. Then, an overlay network of CAP routers, CAP matchers, and CAP executors tries to execute them. This differs from simple service discovery by the automatic execution of a discovered service and by the inclusion of constraints beyond what the service-discovery query for a particular protocol can specify.

Once context information is gathered by specialised services, it needs to be distributed to the interested system components. Event-delivery systems described in the mobile-computing literature have specific event-type hierarchies. For instance, Welling *et al*. [133] uses battery events, hardware add-on events, connection events, and events related to the availability of higher-level services. To avoid overwhelming interested components with notifications about context change, the literature explores the use of tolerance windows registered by recipients [59] and their generalisation to multiple watermark levels [119].

2.5.4. Presenting the context to the user

There is an inherent dichotomy of context information with respect to who is using it and who is creating it. Chapter 3 elaborates on this and how context is handled in the Continuum architecture. Often, context is seen exclusively as something used by software to perform its function. However, to carry on their agendas, mobile users themselves use context, and part of this context is presented to them by the system. In fact, the Continuum architecture identifies this as the main function of the ubiquitous computing middleware. Rudiments of such a view can be found in previous work. Schilit *et al.* [113] builds context awareness into the user interface. One technique is the idea of contextual commands to help the user achieve her goals: e.g., near a coffee machine, the user is presented with specific commands to control it. Another technique is the use of the so-called proximate-selection technique to differentiate the more relevant information: e.g., printers closer to the user are in a larger font. This conveys contextual information to the user to help him define and refine his goals, a system function encountered often in related work. The Guide project [19], which synthesises tourist information for visitors based on their location, can be seen as such an attempt. The Gaia project [42] includes a Context File System that provides virtual folders that are supposed to contain only the files relevant in the current user situation. While filtering files for a virtual folder is dynamic, attaching contextual meta-information to files, such as, location, situation, type of smart space, data validity, or weather, is not.

2.6. Security and privacy in ubiquitous systems

A presentation of the various facets of ubiquitous computing would not be complete without a discussion of the security and privacy issues. The convenience of service access brings a wealth of such issues. Three main areas of security-related issues can be identified for ubiquitous computing: security of end-to-end communication, privacy, and the access, authorisation, and trust. Dealing with all these issues is beyond the scope of the Continuum architecture. Here we survey the issues that the literature considers relevant. Section 2.6.2 points out several security and privacy aspects in Continuum, and relates its role-assertion framework for discovery and service authorisation to the work on which it builds.

2.6.1. User and communication and privacy

The security of the end-to-end communication in ubiquitous computing uses mechanisms proven in the "wired" world, like SSL [117] and TLS [97], to prevent a third party tapping into the wireless network.

In a ubiquitous computing environment, there is more to "security" than just securing the communication. The privacy issues are more severe than in the wired world, and they can be not only a matter of revealing to third parties what data the user is accessing, but also a matter of revealing the user's location and daily routine. In addition, the user might use devices (tablet PCs, stylus scanners, etc.) which bind until reset to the first user that uses them, like in the resurrecting duckling paradigm [118]. In such cases, users will likely choose devices based on the reputation of their manufacturer and possibly that of their administrative domain. Langheinrich discusses the real-world implications of ubiquitous computing [63].

Privacy flaws in ubiquitous computing can have serious consequences that can go as far as physically endangering the user. However, there is a trade-off between convenience and privacy, and there is no such thing as total privacy. With many technologies, the network access provider knows with a certain degree of accuracy the whereabouts of its clients and their identities, e.g., cellular operators are required to be

able to locate a handset with a certain accuracy. Moreover, the presence of an emitting wireless device can be detected easily.

Due to the ad-hoc nature of the interactions in (chaotic) ubiquitous computing, the challenge is how to determine the devices and services the user is comfortable using, from a privacy point of view. An issue that remains is why the user would trust local services. On the web, the P3P platform standardisation effort [21] specifies how to encode such privacy policies in machine-readable format (XML). The use of such encodings would enable ubiquitous-computing software to alert the user when he is about to interact with services or to enter a space that does not respect certain standards.

Wu *et al.* [137] summarises some of the identity-related services that are useful in ubiquitous computing: onetime identifiers, location anonymizers, onetime payment mechanisms, persistent-storage servers, non-repudiation and accountability services, and rendezvous points allowing un-attributable exchange of information. Other examples are reputation services and identity managers. The latter allow the user to present different facets of her identity based on the perceived situation, such as interacting with a vending machine, banking at an ATM, or checking the bus time-table at a bus stop [49]. The privacy routers in the Mist project [80] ensure an entity that knows the user identity either does not know her location at all, or knows it with the granularity desired by the user (e.g., building vs. room [41]).

In ubiquitous computing, the user must be in control of the policies related to her privacy with respect to third parties. This is similar to assumptions made on exchanging data about customers of federated web services in WS-Federation and Liberty Alliance [135, 64, 65]. However, conflicts can arise between user polices and institutional or smart-space policies (e.g., all users in the space must have their identity public, certain users can query who is in the smart space, etc.) [41]. The user could refuse to enter a smart space if he does not agree with the privacy policy. What makes the issue more complex is that smart-space policies can change with the time of day (e.g., office hours vs. weekend) [41], with the users in the space, and with their individual privacy requirements.

Exploring the design space for privacy solutions in ubiquitous computing, Jiang *et al.* [50] looks at the privacy information as a flow, and observes the potential for mismatch between the user perception (e.g., he is in a private room) and the reality (e.g., a video camera is transmitting outside the room). Hence, the collection of privacy-sensitive data should be made explicit. Jiang *et al*. try to formalise a theoretical model for privacy control in information spaces. An information space is defined as a 5-tuple: objects, principals, boundary predicate, the set of operations on objects, and the set of permissions for principals. Their privacy policies, formulated around this abstraction, are controlled when the border of an information space is crossed. For example, in a medical-alert system for seniors, only minimum information is sent to the attending doctor during regular activity, but, once an activity-based boundary is crossed and an emergency occurs, full detail is provided [51]. It is less clear how to capture such boundary crossings.

2.6.2. Trust dissemination and service authorisation

To perform its function of publishing frames and further mediating interactions with ubiquitous services, the Continuum middleware needs access to information about the client and to the data traffic, or at least to the semantic boundaries of the entities in HTTP messages. This implies the user trusts its middleware providers, which is an acceptable assumption for the kind of interactions Continuum mediates and the kind of settings Continuum targets, e.g., a mall, a hotel, a university, an airport, or a commercial street. When this is not acceptable, it is assumed that the owners (employers, businesses, governments, law enforcement agencies, the user, etc) of services and of the data objects will choose security over convenience, and resort to classic the solutions of end-to-end encryption and direct authorisation. Peers could also selectively encrypt information they do not desire to be visible to the middleware. (To avoid man-in-the-middle attacks, data could also be encrypted from the browser to the middleware and from the middleware to services.)

In terms of user privacy, reasonable care has been taken in the implementation of the proof-of-concept prototype to ensure that when Continuum is aware of a certain property of a user, that property is not

inadvertently disclosed to another user by publishing certain tools. For instance, users exchange role assertions through tools, but the tool issuing the assertion and the one receiving it are not paired unless both users enter the same pair of PINs on which they first agree.

As we shall discuss, in the Continuum architecture, the authorisation is left to the target services. Certain credentials and authorisation procedures can be opaque to the middleware. How to control and authorise access to services has been studied extensively in distributed-computing research. Due to the inherently ad-hoc interactions, the user mobility, and the multiple administrative domains, these issues, and in particular the trust issues, are much more complex in ubiquitous computing. (For a cross-disciplinary analysis of the notion of trust, see McKnight *et al.* [77].) English *et al.* observes that in ubiquitous computing, the interactions are more like those in a real society, and, like there, the notion of trust is subjective, depends on the situation, and evolves as more information becomes available [30]. They also note that trust comes from personal observations, from recommendations, and from reputation, a dynamism not generally modeled in the current ubiquitous-computing environments [30].

In distributed systems, a common approach to authorisation has been the use of role-based access control (RBAC [109]). Users are assigned roles. Each role has permissions, and policies are defined around it. However, RBAC models are too static for ubiquitous-computing scenarios that span multiple organisations. This is because the aggregate set of roles is not known in advance, and there is no central organisation to manage the role database.

RBAC models push the complexity of access control into the policies that use the roles as inputs. This motivated much research on how to encode, how to evolve, and where to place such access policies, in other words, how to manage trust. In the area of distributed trust management, Blaze *et al.* [5] proposes to replace identity certificates with assertions that a certain key is trusted for a certain purpose. Local trust relationships through “trust management engines” (PolicyMakers) combine this with local control over the trust relationship

Continuum applies the general notion of role(s) to ubiquitous computing users. The *role assertions*, defined in Section 5.1, represent local credentials, and contain other local information. This framework takes into account the dynamism of the paradigm, and does not depend on design-time roles or on a centralized hierarchy of role-granting authorities. In fact, the solution can leverage current certification systems, such as X.509 [139] and PGP (Pretty Good Privacy) [143] (for a review of certification systems, see Gerck [36]).

From the point of view of the services that make the authorisation decision, the role assertions in Continuum are similar to the assertions proposed by the PolicyMaker. In a way, by issuing a role assertion, an authorised entity extends the rules of the access policy of a service that accepts that assertion. As said at the beginning of this section, such an approach assumes data and interactions for which the benefit of convenience overcomes the risk of potential security breaches. If this happens, it is assumed to be dealt with outside the model (e.g., through law enforcement), and that it results, at most, in a small financial penalty.

There are other research efforts to extend RBAC and trust delegation to ubiquitous computing. The Centaurus project [56] proposes an authorisation framework for a hierarchy of smart spaces. "Security Agents" record delegations and revocations of rights among services. Unlike Continuum, which provisions for peer-to-peer acquisition of role assertions, without special-purpose security servers, Centaurus has limited applicability across non-cooperating administrative domains, due mainly to the need for cooperation between agents for delegation and revocation operations.

The problem of authorising access to services has been considered mostly in self-contained smart spaces. The Aware Home project proposes the use of environment roles [20], which can be organised in hierarchies to simplify their generation. These environmental roles are automatically determined by the system based on environmental conditions monitored by the context widgets [26]. Some of the sample environmental roles used in the Aware Home, like *weekdays* or *business hours* seem too far-fetched to qualify as roles although they can influence the user roles. Others like *mom*, *dad*, *or parent* are closer to

the roles as considered in Continuum (see Section 5.1). Since checking such roles continuously can be expensive, the Aware Home does authorisation on a per-session basis. Unlike Continuum, the Aware Home system, i.e., the user's house, knows the semantics of all these roles and of the rules to generate them. Such a requirement can always be satisfied precisely because the Aware Home is more or less a closed smart space, with a system administrator. Environment roles local to smart spaces could complement the role assertions as defined in Continuum, being just another attribute used to make publication or authorisation decisions.

The Cerberus authentication module [80], a service for Gaia smart spaces [104], starts from another simplifying premise that all the principals are known. Then, a Prolog inference engine works on authentication predicates and context predicates to reason about allowing access. The authentication predicates are provided by a service that authenticates with variable confidence levels. Somewhat similarly, in the Aura project, users can gain one of three levels of access [122]: maximum, where the user is positively authenticated through a physical device; weak, where the user has been authenticated through more error-prone methods such as video recognition; and unauthorised. Earlier, the Gaia project looked at the opposite problem of how to define access-control policies for an access space [107]. Sampemane *et al.* identifies three kinds of roles. *System* roles are assigned when users are created. In Continuum terms, these are a less useful set, since often the provider will not know all its users in advance nor will it create their identities. *Space* roles are based on how the access control is defined. They most closely resemble the roles in Continuum role-assertions. The *Application* roles allow handcrafted software for the particular smart space to specify access policies. In Continuum terms, this kind of role is of interest only to the extent to which computer programs are accessed as services. Also useful is Gaia's notion that the access rights conferred by a role change with the space mode [107]. Several collaborative modes are proposed for an active space: *Individual*, *GroupSupervised*, and *Collaborative*. Some of this work can be leverage by service providers when they implement the authorisation policies for their services.

The authorisation solutions used in smart spaces do not necessarily scale across different smart spaces. Bussard *et. al* [12] takes a different approach to the authorisation in smart spaces. When the user enters a smart space, he obtains a one-time capability from an Access Control Authority (ACA). This capability has similar mathematical properties to electronic cash. Subsequently, he presents it for service access. Essentially, if a user misuses his access rights by reusing the capability, he loses a small amount of money previously deposited as a guarantee of fair use. The service provider has to prove to the bank that the access capability has been used multiple times. This however does not require that the ACA be online at the time of the transaction.

In this chapter, we have presented work related to Continuum from distributed, mobile, and ubiquitous computing. As a general observation, one can notice the need for a framework that promotes interoperability between services, eases incremental deployment, leverages legacy services, and integrates existing and new mechanisms for service and context discovery. It seems that a less heavyweight software abstraction would be more suitable for ubiquitous computing. Such an abstraction should exploit human users' ease of making analogies about the high-level semantics of ubiquitous devices, regardless of the low-level access-protocol differences, and it should let the user be in control of the electronic environment around her. The core function of the ubiquitous system also needs to be (re)defined. The next chapters will address these topics, starting in Chapter 3 with a metaphor for how the user relates to the ubiquitous computing system.

Chapter 3

The role of ubiquitous-computing middleware

This thesis investigates how, by presenting a careful selection of simple tools for users to apply to data, a middleware infrastructure can provide enough structure and coherence for users to understand and exploit familiar and unfamiliar ubiquitous computing environments. The approach stems from the observation that each ubiquitous computing user's task is very fluid. Instead of modelling these tasks, the system should give to the user adequate means to locate and access relevant neighbouring services.

Starting from the metaphor that the Continuum architecture assumes for the interactions between users and ubiquitous services, this chapter advocates a change of what is it currently considered the role of a ubiquitous computing system. To motivate the approach, we first observe the qualitative differences in interacting with a ubiquitous computing system compared to interacting with a desktop computer. A discussion on how the contextual information flows in a ubiquitous computing system follows. We introduce, explain, and motivate the notions of *a priori* and *electronic context*. This chapter advocates that the fundamental function of ubiquitous-computing middleware should be to synthesise electronic context for the user. The Continuum architecture chooses to materialise this electronic context as a set of tools that mobile users apply to their data. Each tool interacts with one or more external services. After defining foundation notions in Continuum, such as ubiquitous services, service providers, user-accessed data objects, and tools, this chapter discusses the how middleware can associate ubiquitous-computing functionality with accessed data. We refine the Continuum architecture into a design where ubiquitous-computing functionality is attached to web pages and the objects embedded in or referenced by them. The chapter ends with details about the network connectivity assumptions that allow the middleware to play its role, and how these assumptions lead to the notions of device attribute and per-user session.

Let us first observe that, when using a desktop computer, the user knows what to do to achieve a particular computing goal she has. This is usually locating and running a dedicated program, previously purchased for that purpose (see Figure 3.1), and interacting with it. The user might also launch additional programs and transfer information among them, through files or a clipboard.

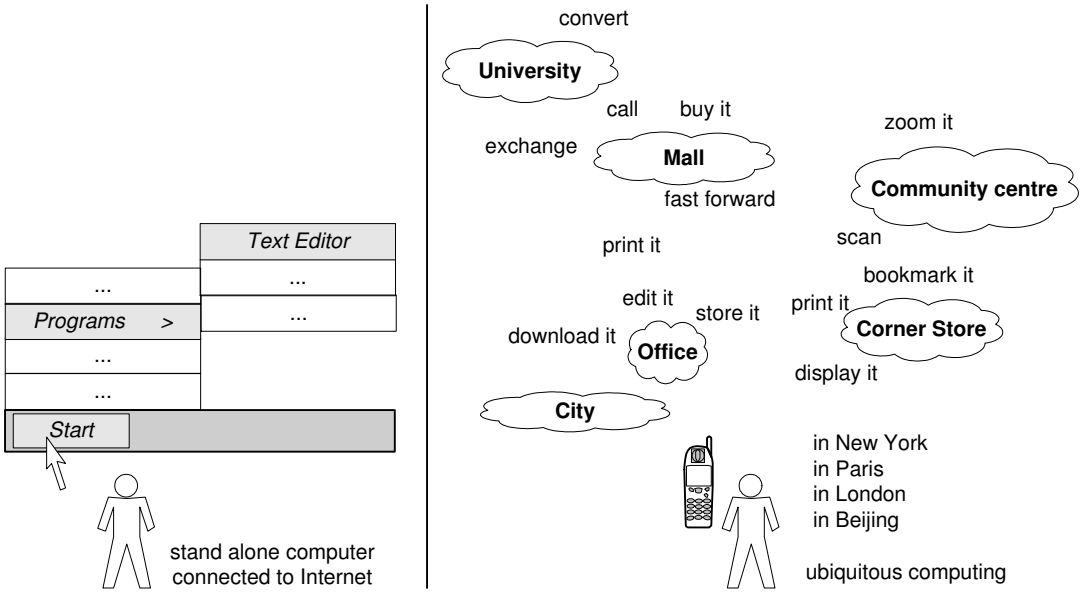

Figure 3.1. Ubiquitous computing interactions versus "traditional" computing

In ubiquitous computing, this model does not work, and an alternative way for the user to start and complete his tasks is not obvious. Most of the visionary scenarios advocating the ubiquitous-computing paradigm imply the user should do nothing at all. The system would guess what the user wants, how it can be done, and what hardware and software components are needed for completion. It might even perform the task on the user's behalf. None of these problems is solved in the general case, and it is probably a naive assumption that computing machinery can guess accurately what a human user wants to do. For desktop computers, such endeavours were only moderately successful, and only for programs with well-defined purposes (e.g., the Office Assistant in the Microsoft Office suite). In ubiquitous computing, the variety and the variability of the possible activities probably make it impossible. For a moment, let us assume success in guessing particular activities in specific application domains. Also, assume the system could be extended dynamically in arbitrary ways, as expertise becomes available. The lack of support for

all possible activities is likely to open a gap between the expectations of the user and the actions of system [31].

The first need of the ubiquitous computing user is to access her data: download it to her personal devices, move it between ubiquitous services, use it to control such services, etc. One can imagine a variety of narrowly scoped operations she might want to perform on data or collections of data she accesses: display, store somewhere else, print, buy, edit, etc. (Figure 3.1, right side). The user will also have to use his personal devices in different jurisdictions, for instance, in New York, Paris, or Beijing, in a corner store and in an infrastructure-rich office, in a mall and in a university, in the city streets but also inside various buildings.

The key observation is that services, devices, and resources accessible in the user's current setting determine, shape, and limit what she can do. Most of these services, devices, and resources will be accessible only from nearby, while others will have more complex access policies. Unlike "traditional" computing, ubiquitous computing envisions tight interleaving between the electronic and the physical space. Many devices, while accessible electronically, have a direct effect on the user's physical world. If in traditional computing, the shell prompt and, later, the *Start* button or the task bar, were clear starting points, in ubiquitous computing, it is not obvious what equivalent to use without imposing an alien view on the paradigm.

It is essential to clarify from both a user and a system perspective how users get to know what they are able to do in a setting, so they can assemble, adjust, or change their immediate goals and tasks. A suitable metaphor is needed for that. By metaphor, we mean how the user relates to the contextual services and what conceptual model he uses to access them, so that he is in control of the interactions without being unnecessarily overwhelmed with choices.

One proposal in related research was the notion of current user task [116]. A task is supposed to follow the user and to be remapped automatically to use external services available in a new setting. The notion has several shortcomings. First, it requires the creator of a task or the user herself to specify the

external services and resources needed. A suitable level of detail is required to facilitate finding replacements. Equating service semantics is a hard problem, and it is unlikely that it can be solved solely at system level, in general, without participation from the user or from software that knows about both service semantics. Second, and harder to overcome, the approach overlooks the fact that lacking a certain service in a new setting or having new service types available might completely reshape the user's tasks in ways that cannot be anticipated by the system. Third, suspending and restarting executions is difficult. Efforts towards saving the user desktop had limited success in most desktop environments (e.g., CDE, Gnome), when the programs are not aware of the process and do not custom-save their state themselves. Automatic suspension and resumption become even more complicated when the desktop session is reinstituted at a different X server with a different resolution, color depth, or set of available fonts.

3.1. Context information flow

A suitable usage metaphor comes together with a modality to capture, generate, and use context. All are crucial to enabling ubiquitous computing on a large scale. Early research in mobile computing has observed that software needs to react and adapt to the dynamism of the environment. Hence, Schilit *et al.* introduced the concept of a *context-aware* mobile application [113].

The *context* is an elusive notion, difficult to represent at a technical level. Dey defines context as any information characterising the situation of a person, place, or object, and that is relevant to the interaction between the user and the software [26]. Schmidt *et al.* defines it as interrelated conditions in which something exists or occurs [115]. For Chen *et al.*, the context is the set of environmental states that determines the behaviour of an application or in which an interesting event occurs [14].

All these definitions consider the context to be an umbrella notion. For Dey *et al.*, the primary context types are the location of the user, his identity, the time, and the activity in progress (what is happening) [25]. In Chapter 2, we survey what elements are used as part of the context in the literature. Schmidt *et al.* considers that the context elements can be split into human factors (such as the user, the social environment, and the tasks) and physical-environment factors (such as the location, the electronic

infrastructure, and the conditions of the physical environment) [115]. Hess *et al.* distinguishes between internal and external context elements. The current user device determines the internal context, while the surroundings determine the external context [42].

Such taxonomies of context elements are not particularly relevant from a system perspective because they do not provide hints on how to architect the system. From such a perspective, the flow of context-information is more important. A key observation is that context information flow is bidirectional. An entity consumes and equally creates context. Moreover, context has instances at different levels.

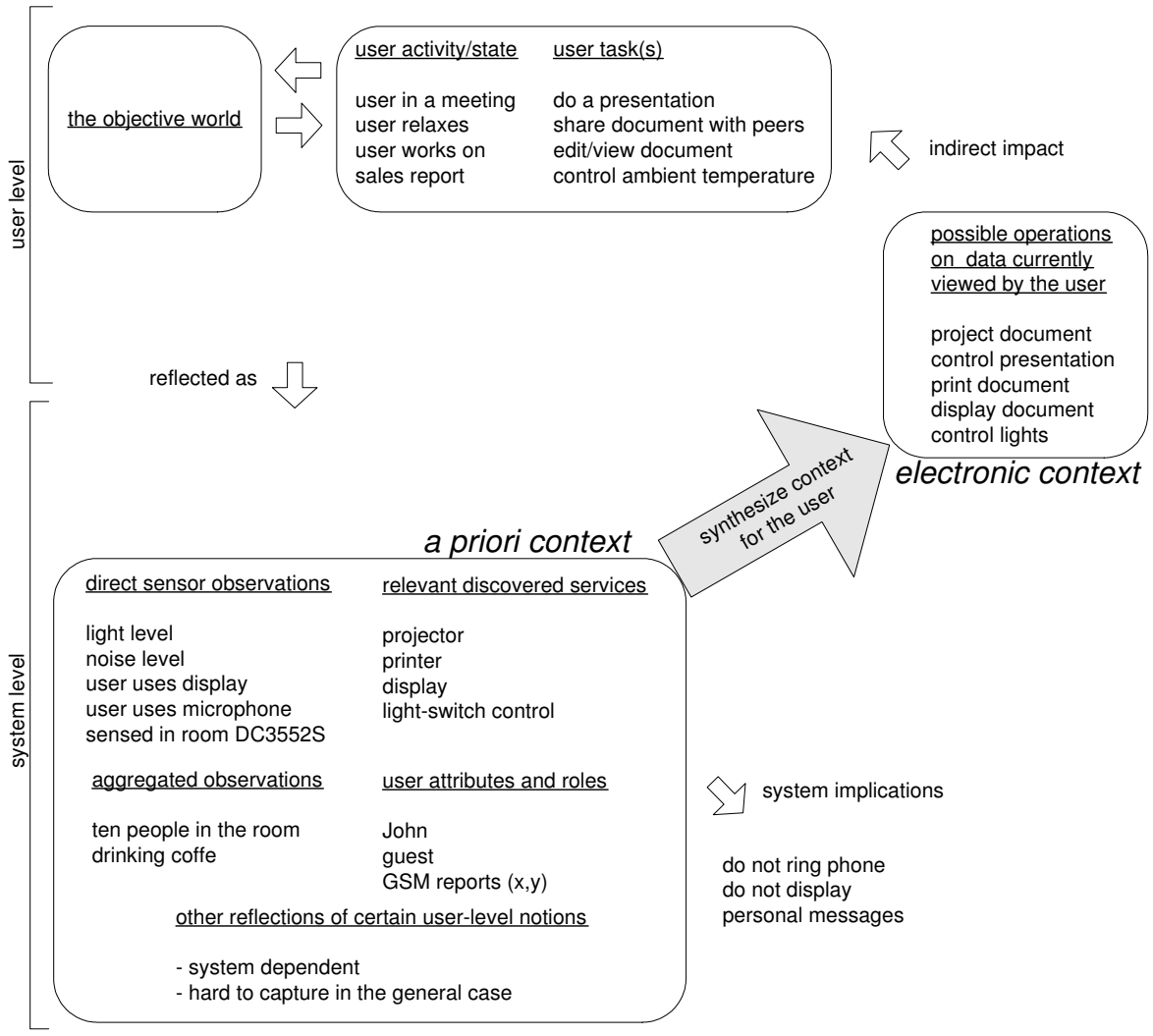

Figure 3.2. The flow of context information in ubiquitous computing

Figure 3.2 shows the flow of context information in a ubiquitous-computing system. The context has materializations at system level and at the level of user perception (user level). The state of things at user level determines the materialization at system level, but conceptually, the latter is a reflection, not a copy of the former.

Only some of the objective world facts (the physical world and the electronic world) and subjective notions (the user's activities, goals, and tasks) are reflected in technical artefacts at system level, and only if they were considered in the initial design of the system. For instance, notions, such as activity, state, or task, have an unbounded set of instances. The impossibility to represent all of them directly is not only a matter of perception (lack of specific information), but a matter of not being able to represent and interpret arbitrary information.

Objective elements are somewhat easier to capture. Regardless of how complicated it may be to measure and determine them, they have clear and concrete definitions. The system finds out about such elements of the context through direct sensor observation, aggregation of such observations, knowledge of the user's attributes and roles (as defined in a subsequent chapter), and discovery of relevant services. The subjective elements of the context influence the objective ones and vice versa. For instance, a light going out might change the user activity from a meeting to something else. Conversely, when a meeting finishes, the user might turn the lights off. More subtly, a change in the user task may change the set of relevant services. To avoid clutter, Figure 3.2 does not provide details about the contextual elements that make up the objective world.

3.2. The *a priori* context

Returning to the general flow of contextual information in Figure 3.2, user-perceived objective and subjective elements of the context are reflected, in one way or another, in artefacts at the system level. We define the *a priori context* as the materialisation of the partial reflection of the "real world" into the ubiquitous-computing system.

Dey observes that working with contextual information in raw form is difficult, hence the need to abstract it to be more easily usable by programs or by the system [26]. To capture this kind of information, there seems to be agreement in the literature on specialised components that monitor and aggregate contextual information from various sources, e.g. sensors. This ensures consistency and avoids recoding functionality.

Devising a general-purpose classification of *a priori* context elements or of a general-purpose mechanism to capture them is beyond the scope of this thesis. Chapter 2 reviewed related work in the area. In this thesis, we refer only as needed to how specific elements of the *a priori* context are handled. For example, location collection and use is left to the discretion of individual frames. (In the current proof-of-concept prototype, the user enters the location directly in a text field displayed by certain frames; the entered value is memorised and subsequently filled in automatically.) In general, a Continuum frame can choose what context elements it uses and it can collect them from arbitrary sources that it locates itself. This is consistent with the decentralisation of the ubiquitous-computing paradigm.

In Chapter 5, we present our contribution to mechanisms that dynamically acquire two elements of the *a priori* context: local service directory URLs and local credentials.

3.3. The electronic context

Once the *a priori* context has been captured, a ubiquitous-computing system can use it to adjust its behaviour and internal state (see Figure 3.2, lower right-hand side). The system becomes what is commonly referred to as an adaptable system. Following contextual input, the system might also manipulate the physical world directly in ways that are beneficial to the user (to avoid clutter, this is not explicitly represented in Figure 3.2). For instance, a hypothetical system might enter a “do not disturb” state and reduce the ambient light.

Much more important for ubiquitous computing is how the system can provide the user with self-help means to realise his goals. We define the *electronic context* as the collection of artefacts and information the system generates expressly to help the user focus and solve his tasks. From the user perspective, this

represents system-synthesised context. The electronic context exists at the border between the system level and the user level. It has a concrete materialization in the system, but it is synthesised exclusively for use by people. In part, the electronic context reflects to the user the contextual information the system has. Chapter 2 shows that rudiments of synthesizing the electronic context can be found in the literature, although they are not typically.

Somewhat similar to our view of *a priori* and electronic context, Chen *et al*. [14] splits the context into active context, which directly influences the behaviour of software, and passive context, which is just presented to the user and has no critical value to the software. The active context is akin to the *a priori* context, but in a less concrete manner and with a more restrictive scope. The passive context as defined by Chen *et al*. represents a subclass of electronic context.

3.4. The Continuum architecture

This thesis suggests that the main function of ubiquitous computing middleware should be to synthesise electronic context for the user, and describes a possible materialization of it, as well as an interaction metaphor that allows the user to exploit it.

Here we define several core notions of Continuum. Figure 3.3 shows the main components of the architecture. The user has a variety of wireless personal devices: a laptop, a PDA, a mobile phone, etc. These devices access remote data over a standard protocol. Figure 3.3 assumes this data is on internet web servers and it is accessed over HTTP, potentially from a within a web browser.

Ubiquitous computing is about access to local external services to manipulate data that the user currently accesses, not about executing independent computations. Situations that ask only for remote computations on servers or for local computations on mobile devices are not of interest to ubiquitous computing. These situations are served very well by remote-access technologies such as the X windowing system, Windows' remote desktop, or *telnet* terminal access, combined with well-understood mobile computing technology such as file hoarding [112].

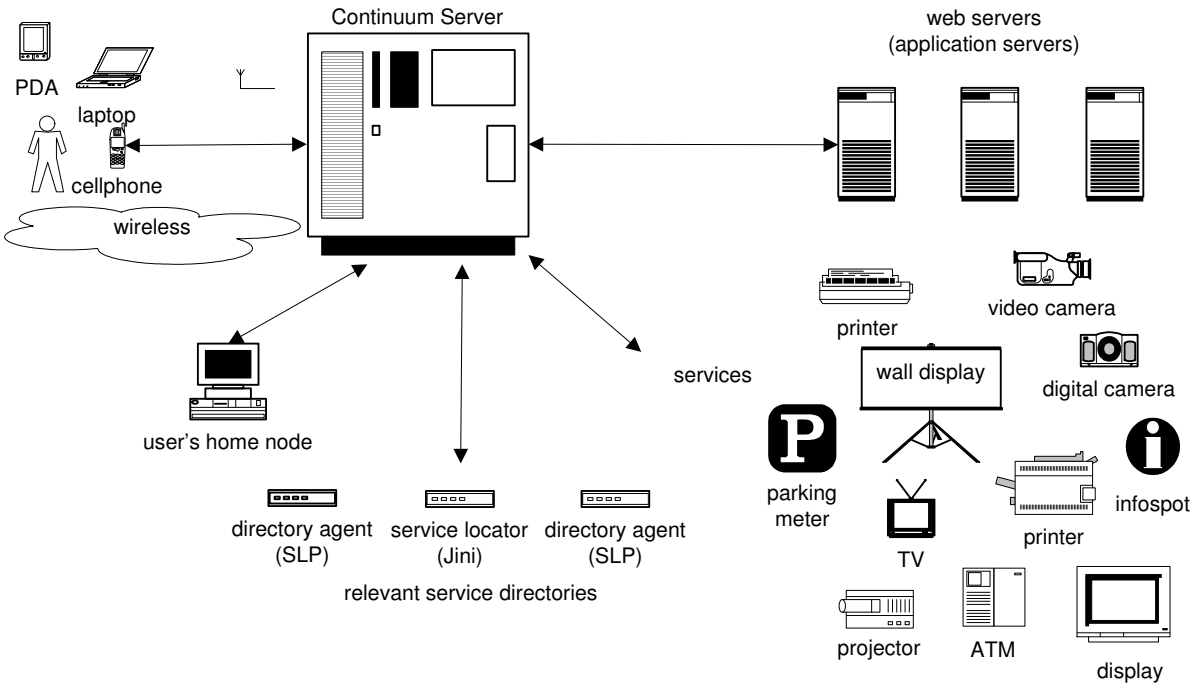

Figure 3.3.The Continuum architecture diagram

Returning to Figure 3.3, the Continuum server is where the middleware runs, mediating user access to contextual services. In Chapter 1, we define a *service* as a piece of functionality that has an impact on or depends upon the state of the physical world surrounding the user. Figure 3.3 shows several examples of services. Services can be offered by a physical device close to the user, or by networked computers (a software service). As mentioned in Chapter 1, a software service normally has particular access and authorisation policies, such as requiring the user to be co-located, to belong to a realm, or to have some trait, such as being a guest or being a participant in a seminar. Globally-accessible services for any user or a set of subscribers are not of particular interest to ubiquitous computing. Their URLs can be obtained easily from global registries, or they can be stored in the user's favourites. Since the user accesses the same instance each time, the software needs access mediation from middleware.

The following aspects characterise a service:

- an access protocol through which it its functionality can be accessed, usually a request-reply protocol, such as lpr, HTTP, or RMI on a Jini object;

- user-level semantics that are given by the functionality or benefit offered to the user;
- a service type that depends on the user-level semantics, but is specified in a way that is particular to the service-discovery protocol used to advertise the service; examples of service type include the SLP templates or the Java interfaces;
- a set of name-value attributes, with semantics that depend on the service type and the discovery protocol;
- an authorization procedure (or no authorization procedure at all); the procedure can be part of the access protocol and depend on the service type;

Accessing a service means using the access protocol, sending a request message, and obtaining a reply message. The request and the reply can contain data, control data (commands), or both; either could be an empty message. The request can have side effects on the service state or the state of the physical world. For comparison only, a Web Service end point [134], for instance, is accessed over SOAP/HTTP with message formats and behaviour (one-way, request reply, solicit response, notification) specified by its WSDL document.

We define an *attribute* as any remarkable information about the user or any property of the user, of her personal device, or of a service. The Continuum middleware considers the user's attributes to be also attributes of the mobile device(s) she uses. For a user's personal devices, examples of attributes are the device IP address, the user location, the user's public key or identity, access credentials, role assertions (defined later), the device type, or the URL of the home node (defined later). We assume the user agrees to disclose some of these attributes to the middleware to mediate interactions better. From the point of view of the middleware, the attributes are part of the *a priori* context. As we shall see in subsequent chapters, the semantics of many of these attributes can often be opaque to the middleware, because they might serve only for the authorisation of service access (discussed in Section 5.2), or be used internally in each frame (defined in Chapter 4). Nevertheless, if the middleware understands the semantics of certain

attributes, it can improve its mediation and especially its synthesis of the electronic context (as discussed further in Section 4.3.1).

In Chapter 1, we also defined *service providers* (SPs) as administrative domains that deploy services, manage and maintain them, and benefit from their deployment. Examples are companies, departments, governments, etc. Their *raison d'être* is to reach users with their services, either for direct benefit, through a pay-per-use model, or for indirect benefits, by attracting users to come and stay in a certain area were they can also purchase goods, like a coffee shop. In the general case, a user cannot reasonably rely on one company to deploy all the services she needs, in all the settings she frequents. Realistically, the user cannot have *a priori* business relationships with an open-ended set of SPs. Most likely, impromptu business interactions will take place. In some areas, there will be no accessible services of a certain type, and this cannot normally be hidden from the user.

Often, each SP will be a different administrative domain, so it can decide where to list its services, in the service directories and protocol(s) of their choice. They can either own service directories, or list services in directories owned by third parties. The service access protocol is at the discretion of the SP, but it is expected that for each service type, a standardized protocol will be used. When SPs elect to use non-standard discovery protocols, or to deploy services of non standard types, some sort of business relationship needs to be established with the middleware providers, so that the latter can mediate access.

Ubiquitous services modify user data, generate data, and act on the physical world, based on user data (e.g., printing) or user commands (e.g., turn out the lights). In an unfamiliar environment, the user will first relate to how to access data needed for her current activity. Only then will she consider what she can do with that data in the current setting.

Users will want to use their personal devices to control all this. Banavar *et al.* [1] notes that user devices are means to perform tasks, and are portals into the user-data space, so mobile-application designers should not see them as mini-desktops or user-managed repositories of software. Newman *et al.* [83] concludes that ubiquitous-computing systems should support the user in arbitrarily assembling the

available resources to accomplish his tasks. Since it is impossible to have programs for each conceivable task the user will ever need to do, general purpose browsers should be used instead [83].

These considerations prompt us to take a data-centric view. A user personal device accesses remote data. These accesses are initiated by the user or by programs on the mobile. Third-party contextual services can manipulate that data. We define a *data object* as a structured document (e.g., an HTML page, a PowerPoint presentation, a photo) and its components and subdivisions (e.g., pictures in a web page, tables in a web page, charts in a PowerPoint presentation, paragraphs, or certain words in any text). Structured documents are more or less the norm on the web.

We assume such accesses are exposed. For instance, in the case of web applications that do not use end-to-end SSL encryption, such accesses are naturally exposed at the level of HTTP requests.

The Continuum architecture considers the ubiquitous computing functionality to be orthogonal to other application logic. The former is essentially attached to the data accessed by the latter. In the case of web applications, the functionality can be added by adding code to the web pages seen by the middleware. Other programs at the mobile would have to have hooks for third-party functionality. For example, the Puppeteer project [67] demonstrates for Microsoft PowerPoint how third-party procedures (hooks) can be called when each structural element of a PowerPoint file is processed at the client. This indicates that the approach of Continuum can be used beyond web applications accessed through a standard web browser. However, only the latter is considered in our design. The resulting client model was defined in Chapter 1

In Continuum, the ubiquitous-computing functionality is represented as operations applied to a data object or a group of data objects. An *operation* moves data between devices and/or services, or transforms data, according to very specialized semantics. Examples of moving data are displaying an object on a nearby projector, acquiring a photo from a digital camera, saving to a disk, or beaming to a peer mobile device. Examples of data transformation are translation, changing the digital format, summarizing, or reducing resolution. One or more external contextual services are normally needed to execute an

operation. The user is aware that, in order to access these external services, she might need credentials, permissions, and/or certain properties acquired dynamically. Section 5.1 discusses these issues further.

The operation is a technical notion. We define a *tool* in a toolbox as the corresponding user-interface artefact for an operation. Chapter 4 proposes a self-contained software unit called a *frame*, which materialises operations at system-level.

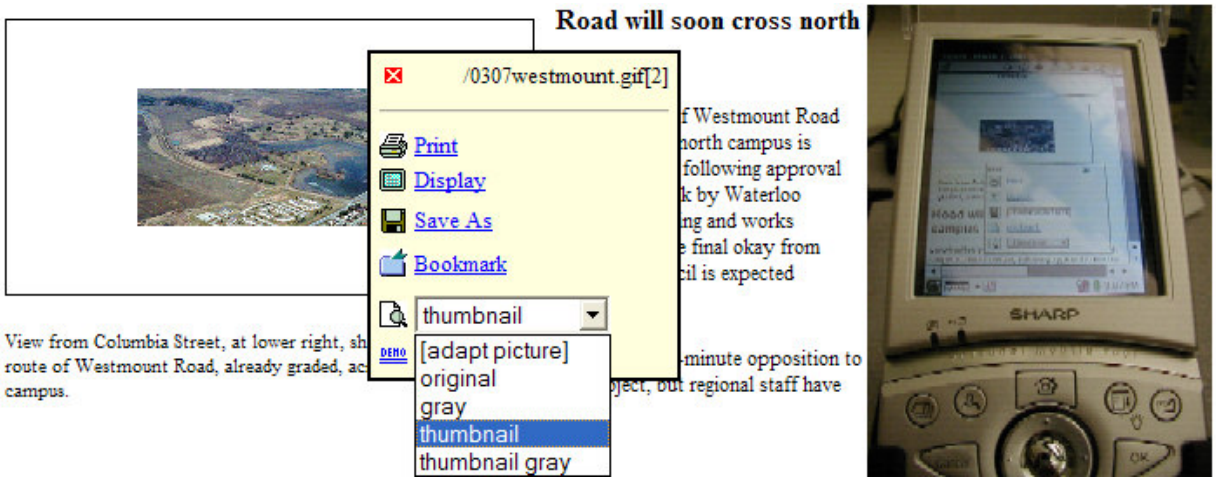

Figure 3.4. Examples of tools in the Continuum prototype

With these definitions, from this point forward, we concentrate on only on the specific case of web applications to be augmented with ubiquitous-computing functionality. Hence, the data objects are HTML pages, the objects embedded into them (e.g. images), and the objects referenced by them. All are accessed over HTTP.

From a user perspective, we adopt a simple metaphor for the ubiquitous computing interactions. The web page viewed by the user in the browser is augmented with dynamically generated toolboxes that contain contextual tools the user can apply to the data in the page.

We define a *hotspot* to be as the artefact that associates a toolbox with the user data to which those tools refer. We have experimented with hotspots that associate tools with the whole web page, hotspots that associate them with individual images embedded in the page, and hotspots that associate them with the links included in the page. We have also considered hotspots that do not attach tools to particular data

objects; some services and operations do not need any input data, but they might generate data (e.g., a digital camera) or otherwise have side effects (e.g., turning off the lights).

As a user-interface artefact, a *hotspot* also represents the means to access a toolbox. Each kind of hotspot conveys differently to what data it refers. How this is conveyed is a UI design issue. In our prototype, the hotspot for the whole web page is an icon in the corner of the page. The hotspot for an image is the image itself, on which the user can click. The hotspot associated with a link is, again, the link itself. The hotspot associated with no tools is added as a special icon inside every other toolbox. Many other kinds of hotspot can be imagined, and several are suggested in Chapter 7. The UI of a hotspot can be different for different "viewers." For instance, if the primary interface with the user is voice, the "hotspot" might be a special keyword pronounced by the user. Well-known properties of hotspots are made available to frame code.

Figure 3.4 gives a brief indication of the user experience with such an interaction metaphor. (The images are from our proof-of-concept prototype). The toolbox associated with the picture was accessed by clicking on the hotspot represented by the image itself. The contextual tools shown allow printing the original image (not the reduced version) on a printer nearby, zooming the image on a screen or display nearby, storing the original image at the user's home node (defined below), and book-marking the image URL, and choosing the fidelity of the transcoded version shown in the browser at the PDA.

Given the "chaotic" and unfamiliar environment, the user needs a starting point, a place to find and save her data and other attributes that apply to several of her personal devices. This role is played by user's *home node*. Conceptually, the home node is related to notions such as the Satchel Personal Web Server [62], the personal mount point in the Context File System (Gaia project) [42], the storage resources in Portolano [32], or the unified inbox in the Mobile People Architecture [74]. We envision that the home node will also play a role in storing attributes and in the execution of the code behind certain tools. (An example of a tool that needs this functionality is discussed in Section 5.1.) More details on the home node follow in subsequent chapters.

The interaction metaphor described in this section is related to what has been proposed in projects such as Speakeasy [83], m-Links [114], and Satchel [62]. As discussed in Chapter 2, from a user perspective, the notion of tool is somewhat similar to tools, task panels, smart tags, and context menus in several office-productivity suites on conventional computers.

3.5. How middleware supports the interaction metaphor

The Continuum middleware enables the interaction metaphor described above by dynamically choosing and instantiating the relevant tools. We use the term *publication* to mean the process of dynamically selecting the set of tools for a particular hotspot, followed by instantiation of the code units behind each of these tools. Portions of this code are added on the fly to the web pages the user visits, and it should obvious for the user how to access the user interface of the published toolboxes. Chapter 4 discusses the publication procedure and the characteristics of the software unit behind a tool. The tools presented to the user are chosen based on the type of hotspot for which they are published, and on other elements of the *a priori* context.

We essentially propose that general-purpose ubiquitous-computing middleware manage, select, and present currently relevant tools to the user, while providing an execution environment for the software behind each tool. The user advances her agenda by using these published tools to apply operations to data objects.

As implied in Figure 3.5, the Continuum server acts as an HTTP proxy, but its functionality goes well beyond altering HTTP traffic. Middleware can perform such a function only if it has access to the data being browsed. We have explained in the related work section why this is a reasonable assumption.

The assumption is that software developers design small units of code (the frames) to improve users' experience by easing and customizing the access to services deployed by third-party SPs. Middleware providers deploy an overlay infrastructure to manage this software. The middleware also executes portions of it.

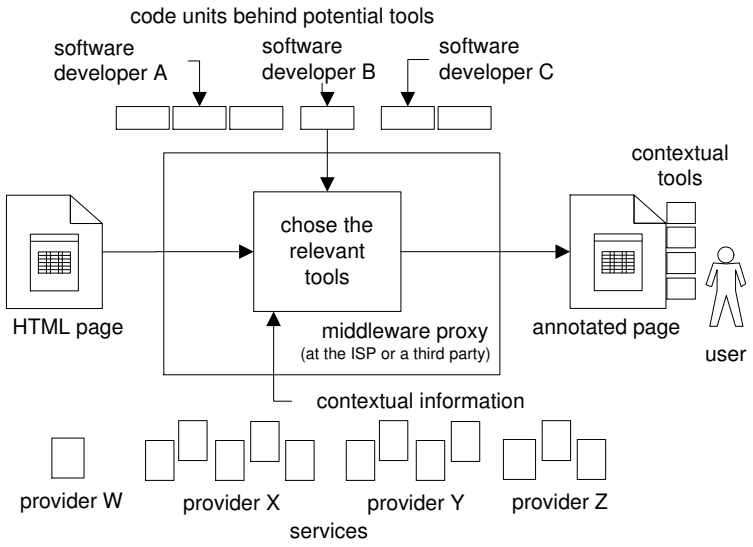

Figure 3.5. The dynamics of a ubiquitous computing system

There are several aspects pertaining to the general notion of electronic context. First, it is an approximation. If an oracle were available, one could imagine synthesising the minimal context that best helps the user to pursue only her current endeavours. Without such an oracle, this is not possible, so the electronic context can only be an approximation.

Moreover, the *a priori* context itself is inherently an approximation, since its elements are simplified reflections of a subset of the contextual information existing outside the system. The system might not yet know of some services or service directories, or of all the attributes and preferences of the user. Engines for situation inference usually work on heuristics, so they are occasionally wrong. Many characteristics of the mobile environment are statistical measures (throughput, handoff-dropping probability), so they are affected by the interval on which they are estimated, by the sampling rate and by the way the information is gathered, such as using special out-of-band packets or using regular data packets. Sensor readings might be inaccurate or have different levels of accuracy, such as expressing a latitude-longitude location as a room number. For example, to quantify this reality, the Contextual Information Service (Aura project) [55] supports meta-attributes such as accuracy, freshness, confidence, and sample interval for contextual information.

We consider that the user should be aware that the electronic context is an approximation, since users know they are in an unfamiliar environment, about which not all information is available. The user might be required to control the expansion of the electronic context actively by looking for and acquiring local information that can improve the publication heuristics.

A second notable aspect is that the user experience with the system is influenced by how often it updates or regenerates the electronic context. As the user moves or the environment changes, even the ideal electronic context degrades if it is not refreshed.

Third, by interacting with the artefacts of the electronic context, the user will modify the world and, hence, eventually the *a priori* context. An arrow in the upper-right corner of Figure 3.2 indicates this feedback loop.

Summarizing, a toolbox might contain more tools than necessary, tools for services the user chooses never to interact with, or tools that become invalid after publication because the user moves and conditions change. More often, the set of published tools will be smaller than ideal.

3.6. Connectivity assumptions in Continuum

Essentially, the middleware assumes the functions of putting user-viewed data into context. Chapter 4 discusses the issues associated with this synthesis. Meanwhile, we present a set of assumptions that allow the middleware to perform its function.

The user's personal devices have to be connected to the internet. Given current technological trends (3G, WiFi, Bluetooth), ubiquitous-computing interactions will often happen over medium-lived network connectivity. Some of the current technologies have rather high connection-establishment overhead. (For example, it can take up to 2.56 seconds to establish a Bluetooth connection, if the two devices know about each other's MAC address and clock value, with an additional 10 seconds or more if they first have to discover each other [75].) When users interact with services in a certain area, speed is low (walking pace), so connectivity can be maintained. At higher speeds, the user is probably in a vehicle, so the vehicle can act as a base station that connects several personal devices. These can connect to the vehicle hub over any

technology. The car could connect to the internet over a technology that performs hand-offs at sub-IP layers, such as, GSM.

In spite of the assumption of medium-lived connections, the middleware, the other software components, and the ubiquitous services still need to provision for unexpected connectivity drops. These might be due to switching the ISP (e.g., switching from using the wireless access offered by a coffee shop to using the wireless access offered by the university), and/or switching the access technology (e.g., from GSM to WiFi).

Medium-lived connectivity allows the middleware server to maintain soft sessions to serve its clients better. A Continuum session caches attributes of the user device with which the session is associated. Consistent with the best-effort approach in Continuum, the attribute pool is merely a cache. This pool is populated with attributes observed by the middleware in the communication to/from the client.

As a design choice, the main copy of an attribute can be stored at the browser, in a cookie. Attributes that refer to more than one device the user owns can have the main copy at the home node. Devising a context- and attribute-collection service, a context-aggregation service, a user-location service, or a location-conversion service are beyond the scope of this thesis.

A Continuum session also caches the URLs of known service directories that might contain services currently relevant to the user of the device. Section 5.1, provides details of how the mobile and then the middleware acquire such information and Chapter 6 discusses how sessions are implemented in the Continuum prototype.

Because of physically moving, the user might have to change middleware servers, but session hand-off is not required because Continuum sessions are soft, so the associated information can be collected again. Nevertheless, to maintain the level of service, one could imagine implementing session hand-off. Related research has studied proxy hand-off in the context of streaming media, e.g. Mobiware [13], or has avoided the need for it by modifying mobile IP to always route packets through a particular node [142]. Unless prior agreements exist between parties on authorisation, data formats, and protocols to be used by

the middleware, middleware handoff can be problematic when the device moves beyond the administrative domain of the middleware, into the service area of a different middleware provider.

Summarising, starting from the flow of context in ubiquitous computing, this chapter introduces the notions of *a priori* and electronic contexts, and argued that the main role of a ubiquitous computing middleware should be to synthesise the electronic context dynamically for the user. In Continuum, this context is materialised as a set of published tools for applying operations to data accessed directly by the user, or by other application functionality (e.g., a sales chart, a presentation, a vacation picture, a web page). These tools make the ubiquitous-computing functionality orthogonal to other application functionality.

The tools are user-interface artefacts. In the next chapter, we introduce the Continuum frame as the software unit behind a tool, and show what functionality a frame needs to implement in the general case and how the middleware does frame selection and publication.

Chapter 4

Frames: A software building block

Several elements contribute to enabling ubiquitous computing: ubiquitous deployment of networked electronic devices, ubiquitous wireless and wired internet access, portable user devices, standardized protocols and formats, and, more significantly, software that allows the user to make sense of and use services with arbitrary semantics. While most of these elements are a reality, this is not true of the software, especially in the case of ad-hoc interactions across administrative and technological domains. This chapter explores a software abstraction for ubiquitous computing, which enables independent developers to contribute to the experience of a ubiquitous-computing user. We discuss the need for incentives to write such software, and the need to manage it and to recover development expenses.

The traditional notion of computer program does not fit well in ubiquitous computing, so we advocate a simpler software abstraction called a *frame* that imposes structure for both the user and the developer. Starting from the unique characteristics of the ubiquitous computing environments and the design goals outlined in the previous chapters, we investigate the necessary functionalities of frames, and discuss mechanisms needed at the middleware level to manage and publish the frames. Chapter 6 gives further details on the implementation of frames within a proof-of-concept prototype.

4.1. Why another software abstraction?

We begin by justifying what makes existing software abstractions less suitable for ubiquitous computing. Figure 3.1 shows how the user traditionally starts interactions with the programs on a desktop computer, and how much more complex the situation is for ubiquitous computing. From a single computer to networked computers and then distributed systems, the notion of computer program as perceived by the user has not changed. Often, the distribution is hidden by making the distributed system behave like one computer with its attached peripherals. Even with the wide adoption of the web and web

applications, the user essentially visits a URL and interacts over the web with a server application to achieve something.

The most popular approach to ubiquitous computing, the smart space, still employs the notion of program to a certain extent. By computer program, we mean a rather large piece of software that executes at one or more computational nodes and it is designed for a certain complex goal, such as editing a text document, retouching a photo, etc. Essentially, related research treats a smart space as a computing system running a meta-operating system (e.g., iROS [53]). This is facilitated by the rather tightly knit environment it represents, having device semantics and purposes already known at design time of the application, of the software for the space, and of client-device software.

Consider now the traditional idea of an application, in the sense of all the infrastructure and software deployed to help one or more users solve a real-world problem in a particular domain. The user will interact simultaneously with multiple such ubiquitous-computing applications. She might be tracking a parcel, using a warehouse-monitoring application, finding her way through the building with the help of a navigation system, and accessing her calendar, and then suddenly decide to buy a beverage from a vending machine that happened to be along the way.

Currently, special-purpose infrastructure and software need to be deployed and configured for such an application. In fact, a smart space itself is an application, since it has special purpose, and its hardware infrastructure is specifically deployed in a certain location to serve that purpose.

Often, users in a smart space can use only the application for which the smart space has been designed. It would be an improvement to facilitate the use of the same infrastructure for additional applications, and to allow the user to use the deployed infrastructure for other activities, not foreseen during the design of the original application.

Since different architects will design the different ubiquitous computing applications, it is important for the developer to understand where the boundaries of each are and to comprehend how they might or should interact. On the other hand, the architect of a ubiquitous-computing application cannot guess, at

design time, with what other applications the user will have to interact, and it is not clear how to devise or characterise the boundaries among them because these applications are interleaved with the user's physical environment. This means they will interact with each other not only over software interfaces but also indirectly by affecting the physical world. This makes things more complicated than in distributed computing.

Current ubiquitous-computing applications assume certain components are running on the user's personal devices to allow them to be part of the application. It is likely that, in the general case, user devices will come to interact with smart spaces where the assumptions about the discovery protocols, communication protocols, authorisation protocols, service types, service attributes and properties, and meta-operating system may differ from those made by the built-in software on the mobile device. The opposite problem also exists. When entering a smart space, users might carry devices of types unknown to the designers of the space. Moreover, a user might need to interact simultaneously with multiple smart spaces, or with services and a smart space belonging to different administrative domains.

All these reasons suggest smaller software abstractions are more suitable for ubiquitous computing. Unlike a generic distributed-computing component, say, an Enterprise JavaBean, the software units we are proposing have specific functionality stemming from the particular characteristics of ubiquitous computing.

The users need non-disruptive, functional, and interoperable software. When presented with a set of services, a user might want interactions that seem natural to him, but the system might deem them not applicable to the current situation, or the application designer might not have considered them. This issue exists of course with any computer program, but it is more of a problem in ubiquitous computing because of the closer interleaving with the user's physical space.

Given the continuous evolution of the kinds of available services, it is unrealistic to assume universally accepted standards that would make the problem go away. Such hypothetical standards either would settle for the smallest common denominator making them ineffective, or would be out of date,

since previous experience shows standards bodies are often lagging behind in standardising new technologies.

The degree of ad-hoc interactions in ubiquitous computing is higher that in distributed computing. Eventually, software on the mobile device that expects a certain service-interface name and signature from a peer software unit might have to interact with software with a similar interface but, say, a different method name. Potentially, they could interact through an adaptor, but in practice, they will not, because, even if the signatures of their methods are similar, there is no way for the software to know that the two interfaces do have similar semantics. Of course, one can claim that new software interfaces or updates on the user devices will solve the problem, but, in the general-case, the set of such interfaces is large and so is the set of administrative domains that have control over them.

To address the problem of matching high-level semantics in ubiquitous computing, it seems more useful to have software units that interface directly with the user. This is one of the important contributions of the notion of frame. It increases interoperability, by leveraging the ability of human users to smooth out semantic anomalies and differences. Indeed, users are much better at figuring out similar semantics and finding creative uses for services than software. Consider the example of a button to play a movie. An able adult will have no problem if the button is labelled “start,” instead of “play,” but a software component expecting a method named “play” will break if a different name is used. Eventually, the semantic web may alleviate these issues.

A software unit also has to take into consideration that ubiquitous computing takes place across administrative domains. This makes SPs a poor choice to offer such software units. They have no incentive to encourage the use of similar and potentially more suitable services, provided by competitors. This indicates that a third party, different from the providers of the service, should be involved. Such a third party would be a middleware provider. ISPs could act in this capacity. Of course, the authorisation of access remains a responsibility of the administrative domain.

The often neglected issue of how ubiquitous computing software is licensed (who pays for software) is also quite important, ensuring that there is some incentive to write software to improve the user experience. This comes with the matching issue of how the user identifies what software unit brings her a specific benefit, and who provided it. When purchasing a device, just as when purchasing a pen, the user has a clear idea of the benefit. The manufacturer and the merchant also know how they recover their expenses. When the user is accessing a certain service, the source of benefit to the SP is also clear. It can recover expenses by charging for the access, through a flat fee plan, or even from the collateral benefits of users returning to a certain space (e.g., a free electronic service offered by a shop lures customers to return to the shop). An organisation that fits one of its rooms with electronic equipment and software to turn it into a special-purpose ubiquitous computing space, such as a meeting room, also has a clear notion of the value that space will bring. In contrast, it is not clear who should pay for software that allows access across administrative domains.

All these considerations suggest that the first-class object at the system level should be a self-contained unit of software that interfaces directly with the user, and has the function of implementing a certain operation using external services. As discussed in the previous chapter, the function of the middleware system becomes the dynamic selection of the units it instantiates and presents to the user in a certain situation. The use of such a self-contained unit would simplify the development, by reducing the complexity for the developer, without unduly restricting the opportunities that arise from ubiquity and integration with the real world. Such an approach also moves away from trying to capture user activity as a first-class object, which is a notion too fluid for the system to infer. Instead of guessing this, the task of the middleware becomes to present to the user a materialization of the electronic context that will enable her exploitation of the electronic surroundings to further her agenda.

4.2. Anatomy of a frame

Continuum frames are such self-contained units. Since we are primarily concerned with reusable aspects of this design, in the reminder of this section, we keep the description independent of particular

programming languages and infrastructural elements. Subsequently, especially in Chapter 6, we describe how this design was implemented in our proof-of-concept prototype.

Before proceeding, we note that the Continuum frame logic is typically split between an applet, running at the device (usually within a web browser), and a servlet, running on a runtime provided by the middleware. To avoid keeping state at the middleware, applets are usually stateful, while servlets are stateless and are instantiated on demand. An individual frame may be implemented as a stand-alone applet, as an applet driving external contextual services, as an applet communicating with a stand-alone servlet, or as an applet communicating with a servlet that drives external services. The last design will probably be used most often. Later in this chapter, we identify what part of the frame functionality the frame applet and the frame servlet typically implement, but first let us identify what kind of functionality a frame, as a whole, needs to implement.

First, consider the *Display* frame, a frame implemented in the Continuum prototype. The user views a map on her PDA. When she finds that a map from the web is not rendered acceptably on the PDA screen, she clicks through the map to access the toolbox. Then, she chooses the *Display* tool, to magnify the small map on a nearby public-use display. Figure 4.1 gives an idea of the sequence of windows as the interaction proceeds. The first step of choosing the frame from the toolbox is not shown to avoid clutter.

The interaction with the frame starts with a dialog box that asks the user about his current location. The prototype follows the very simplistic approach of having the user input his location. In a more elaborate prototype, this would be obtained differently. As outlined in Chapter 2, obtaining the location depends on the network access technology. For instance, in the picture, the PDA connects to the internet through its cradle, and potentially the PC to which is currently connected knows where it is located. The same PDA can also access the internet over a CF wireless interface, in which case the access point might know the location. Other cards might enable other internet access, such as cellular. In Chapter 2, we discussed why obtaining and using location are hard problems. Solving them is beyond the scope of this thesis. Returning to our example, if the location has already been introduced during interaction with other

Continuum frames, its value is automatically filled in, and the user can correct it, if it is no longer up to date. The frame needs this information to locate the displays available nearby.

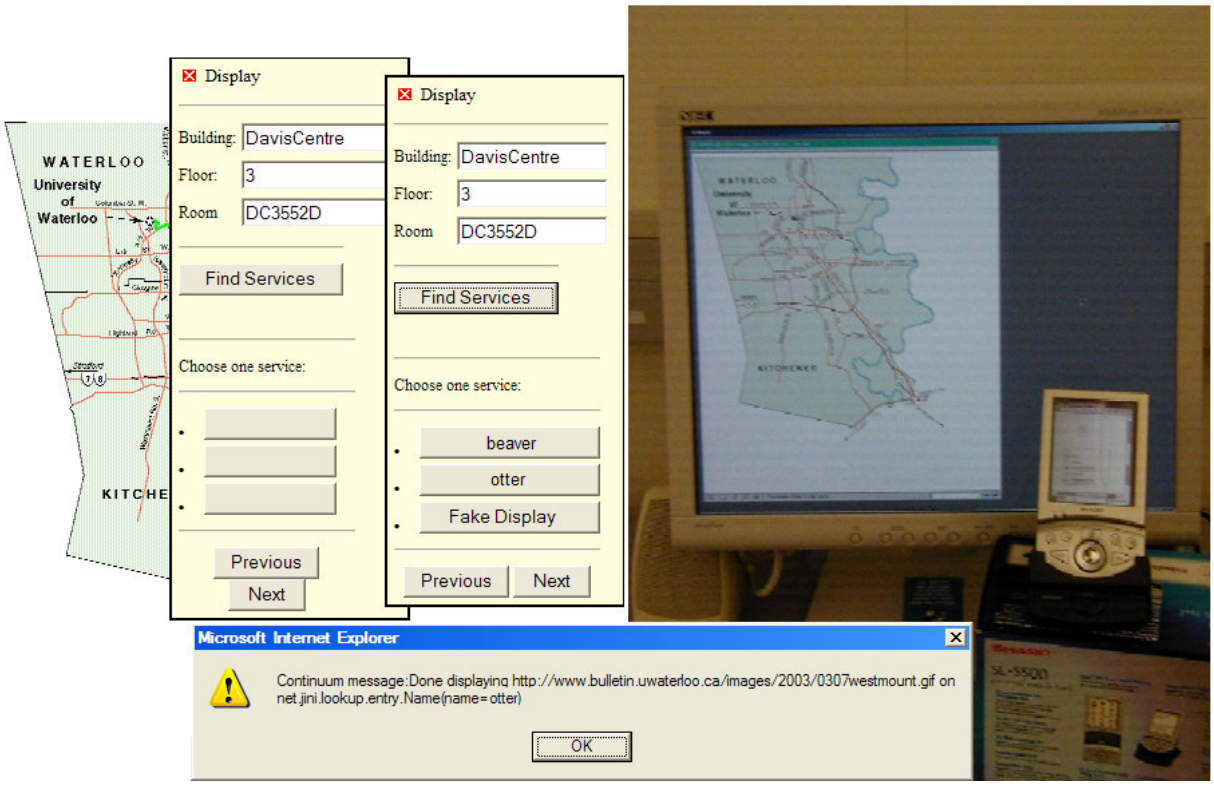

Figure 4.1. The user magnifies the map from the PDA screen on a nearby display

By clicking on the *Find Services* button, the user instructs the frame to look for nearby displays. The details are discussed in subsequent chapters. Three services are found (two are the screens of real computers, the third is a simulated display service used for debugging). The user chooses the target display and a message box confirms that the map has been displayed. At this point, the PDA user is free to continue browsing. The map stays on the nearby display until the user closes the window on that screen. (Alternative functionality can also be imagined. The frame could behave like a modal dialog box and the user would interact with the zoomed map only from the PDA.)

To enable these interactions, the *Display* frame first had to be published. Then, it had to interact with the user, look for external displays in service directories in the user's area, and drive those displays, using

their respective protocols. The data transferred by the frame to the target display has to be converted to a format understood by the display service.

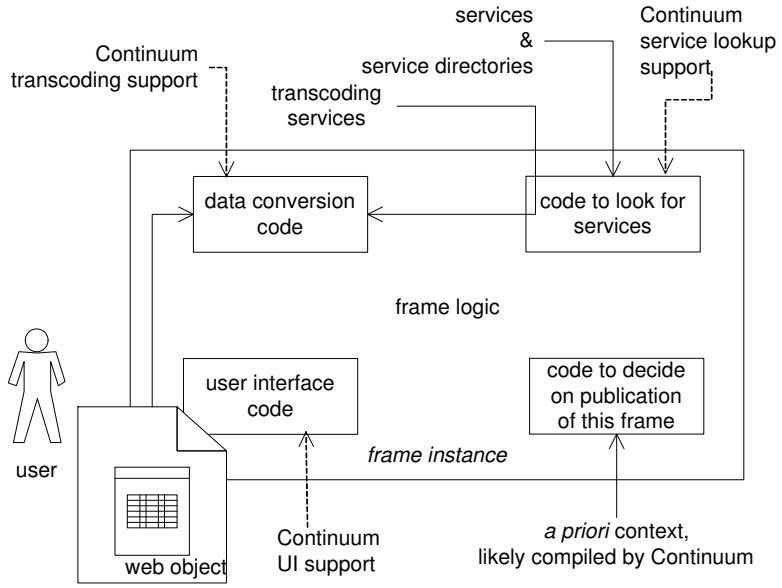

Figure 4.2. Frame functional properties

Figure 4.2 illustrates the functional properties of a frame. The code that implements the operation–the frame logic– combines the functionality of the external services, interacts with those services (following their respective access protocols), drives the process of authorizing the user to access the service, and controls the user-interface interaction. Note in the lower right-hand side corner that, as we see later in this chapter, a frame can use contextual information to decide on its own publication. This assumes there are standards of how the contextual information is encoded and how it can be accessed by frames. Currently there are no such standards in place, but there is a large body of research on the issue, as reviewed in Chapter 2. This thesis does not concern itself devising such standards and protocols. We note that it is likely there will be several competing standards, so different administrative domains could pick any of them, as is the case with service-discovery protocols. Consequently, a frame could support fully or partly one or many of such standards. If a frame cannot obtain required contextual information, it can veto its own publication.

4.2.1. Interfacing with the user

Implementing the frame behind a tool starts with designing and coding the user interface (UI). For each frame, a certain UI "real estate" is allocated in the toolbox during the publishing process.

The frame UI conveys the semantics of the frame to the user. While the interfaces for frames with similar functionality do not have to be the same, they should contain enough cues so that users understand the purpose of the frames and the side effects of interacting with them. In simple cases, the general semantics of a frame can be conveyed through as little as an entry in a menu. Finding good labels can sometimes be challenging. Frames could also have arbitrarily sophisticated interfaces, including a voice interface if the client device supports it.

After the publication, when the user chooses to interact with a frame, it may gather additional input from him. This can be additional contextual information not available directly from the browser/infrastructure, or other parameters needed to execute the frame.

If the frame is using external services, it can be hard to convey unequivocally to the user which electronic services are involved in the execution of a frame, especially given the small screens of mobile devices. (When a voice interface is used, brevity is also desired.) A *Print* frame might have to indicate that the printout will come from a printer on a different floor, not from the printer in front of the user. Given the decentralization and different ownership, the contextual services might be characterised inconsistently or sparingly by their attributes. Network addresses are usually meaningless to users, while location can be inaccurate or too coarse (for instance, all SLP [100] printers in the same room might have the same location in the *printer-location* attribute). In the context of transcoding web content for display on small screens, the m-Links project [114] attempts to compile short, meaningful, and unique labels for data that has been detected as being interpretable as links, e.g., telephone numbers. M-links tries to overcome the fact that web pages often have irrelevant anchor text (e.g., "click here"), obscure URLs and poor HTML titles. Ideas from this research might be usable in the ubiquitous-computing context. However, this is a per-frame issue and we do not deal with it directly in our prototype. When services require unique

identification, we use the less user-friendly alternative of concatenating the service URLs and other service properties.

Most often, the user interface will be implemented by the frame applet running at the user's device. Its design and coding can benefit from special-purpose libraries and templates. For web applications, the applets will run in an internet browser, and typically, they will be implemented in a combination of HTML and JavaScript. They can assume a certain runtime environment within the browser, created automatically in the web page by the middleware. The frame developer can also choose from a few HTML+Javascript templates (we return to this in the next chapter).

Frames could also have their UI generated or adjusted automatically when they are published [43, 91]. In the future, the frame could also capitalise on innovative concepts such as steerable interfaces [90]. We have not explored these alternatives further. Given the variability of the unbounded set of service semantics, it is unlikely that automatic UI-generation will be possible for all frames, so some will still have to be designed manually.

4.2.2. Data conversion, versioning, and transcoding

Having discussed the frame UI issues, we move to analyse the implications of the data-centric design. Often, services with identical high-level semantics accept and generate different data types. Even widely deployed, standardized, services like printers can accept Postscript, a proprietary language, or both. To complicate things further, different formats can have the same apparent fidelity to the user (e.g., Postscript and PDF) while others do not (e.g., JPEG and GIF).

In short, data objects have versions. Versioning is tightly interwoven with the semantics of the operation implemented by the frame, and consequently it rests with the frame user interface to convey any specifics to the user. Unlike the notion of copy, customarily used in replication, a version at a service can be different at binary level from the original version, without necessarily being stale. The notion of fidelity [85], defined as the degree to which a copy of the data presented for use matches a reference copy, is not always expressive enough.

There are several aspects of the equivalence between two versions: qualitative equivalence, the degree of content equivalence, the potential for conversion, and even the semantic degree of equivalence. An example of qualitatively equivalent files is two files in different file formats that are indistinguishable when rendered on one device, although they may look different on another, such as raster and vector images. The degree of content equivalence refers to versions that can be obtained one from the other through some transformation that reduces the information, such as a summary of the text, or a thumbnail of an image. The potential for conversion refers to the possibility of converting one version into the other. For instance, the printed version of a GIF cannot be converted back to the original (at least, not without human intervention). An example where the semantic degree of equivalence is relevant is a legal contract, translated on the fly from English to French.

In general, a frame directs transcoding itself, to provide each of the services it drives with a version it can accept. This transcoding is tied tightly to the particular frame semantics, and the developer cannot avoid having to control how conversions take place. For instance, does a frame that prints nearby print the outline of the document, does it print what the user sees (potentially a poor quality version), or does it print the original? The frame has to convey the semantics to the user and fetch and transcode the appropriate version. Adapting data objects in transit to the client has been a topic of extensive research since Brooks *et al.* [9] first suggested that web proxies should contribute to improving the user experience by filtering HTTP requests and responses.

When a system component makes an adaptation decision, it is expected to make the user aware of the alteration, if this is relevant to her. For instance, a user would be interested to know if the legal document he is seeing is a certified translation or not, or if it has the same legal power as the original. Signalling transcoding alterations is the responsibility of the frame that directed the transformation. Sometimes, an alteration made by the middleware can be signalled implicitly. For instance, to save bandwidth, the prototype middleware replaces images going to the browser with blank GIFs. This is signalled by the

publication of an *ImageAdaptation* frame that can further adjust the appearance of the image, with a choice of grey scale or colour, thumbnail size or full size.

A frame might be associated with a non-identical version of a master copy, e.g. the thumbnail of an image. If the logic of the frame modifies that version after interaction with the user or with one or more contextual services, the changes need to be propagated back to the master copy. This is a difficult problem given the different levels of fidelity. Solutions for maintaining consistency, like Coda [112], can be used only when an exact copy of the original is modified. Lara *et al.* [67] shows how to incorporate fidelity into existing replication protocols in an orthogonal fashion. Their implementation keeps track of the consistency and fidelity of structural elements of a document, such as paragraphs, images, or whole pages, and shows how changes can be propagated back to the original copy. Frames that modify the data version they are associated with and then propagate changes back to a master copy have not been investigated further in this thesis.

To summarise, a general frame will manipulate several versions of a data object, convert them, and exchange them with different services. It might generate data objects and it might even have to propagate changes back to a master version. At the frame code level, this multitude of versions brings the issue of naming and characterising the various versions. This is not of concern to the middleware infrastructure, but to individual frames. For example, the applet of the *ImageAdaptation* frame in Continuum appends name-value pairs after the query sign in the URL of the image to be displayed. These requests are intercepted by Continuum and sent to the corresponding frame servlet, which manipulates the original accordingly and returns the resulting version.

Controlling conversion does not necessarily mean the developer implements the actual conversion inside the frame. The frame can locate transcoding services in the same way it locates other services. For select formats, it might also use middleware-provided conversion functionality. The lack of local availability of certain transcoding services is signalled to the user the same way as the lack of other external services the frame needs. If the lack is known at publication time, the frame is not published.

4.2.3. Service discovery

This brings us to finding all the external services a frame needs. Current ubiquitous-computing systems are handcrafted based on a set of simplifying assumptions. For example, Ponnekanti *et al.* [91] reports that with the iRoom, they felt no need to use existing discovery protocols, because they were served well by the event heap at the core of their infrastructure. In Chapter 2, we briefly review the variety of standard service discovery protocols, with respect to their immediate purpose, to how they ensure interoperability between services, to the discovery mechanisms used, and to what attributes they enforce.

All these considerations show that equating between service types is a hard problem that, in general, it cannot be delegated to the system or to the middleware. First, the set of service types is an ever-expanding set. Second, a commonly agreed description language expressing what kinds of services the frame uses would need to be devised. In some cases, this can be done—an example is the XML description of a task in the Aura project [116]. In general, for an arbitrary service type, an arbitrary set of conditions on the service attributes might make the difference between a suitable and an unsuitable service. One cannot predict at language-design time the complexity of the language constructs one may need, or, say the complete XML schema. Since a developer can always imagine more esoteric service-equivalence tests, a description language will have to be as rich and expressive as a general-purpose programming language.

Third, middleware would have to match service types and attributes across service discovery protocols. Even if the software interface for services were simplified to a bare minimum, e.g., HTTP, or the minimal service interfaces devised in the Speakeasy project [29], it is difficult to do such matching without exact knowledge of the frame purpose and semantics.

Moreover, automated matching can yield imperfect results. For example, a projector might be matched as a display replacement to show a confidential document. Automated matching can also overlook peculiar matches suitable enough for a frame with particular semantics. For instance, a fast printer could sometimes be sufficient as a display replacement for a frame that needs to communicate to the user a large quantity of information. Automated matching also needs to take into account the

semantics of the frame. Consider, for example, the differences among the semantics of the following three frames: *PrintInTheRoom*, *NextDayDeliveryPrints*, and *DepartmentalPrinting*. All require a printer, potentially the same kind of printer, but there are significant differences in the exact query to locate that printer.

This shows how service use and service discovery are connected, both relying on frame semantics and specific properties of the potential target services. Consequently, the frame code must know how to obtain the services it requires, or at least how to issue discovery queries that can locate these services. A frame will be able to look only for the service types for which it was built.

If frames take charge of issuing discovery queries, one avoids the need for a universally agreed discovery protocol, and an attribute and service-type hierarchy. Ongoing efforts towards devising ontologies and creating the semantic web illustrate how hard it is to devise a generally agreed way to describe entities with an expressiveness level that would satisfy all possible future queries.

Shifting the problem to the frame level simplifies it significantly. Each frame commits to having knowledge of the semantics of a specific subset of discovery protocols. It is then capable of formulating discovery queries for service alternatives over any of these protocols, depending on what service directories are known to the middleware. In Chapter 5, we discuss an infrastructure mechanism to become aware of service directories relevant to the user. For the protocols known to the middleware, it can provide query facilities to frames directly.

Regardless of how robust a frame is in discovering and using similar services advertised over different discovery protocols, there may be situations where this fails. For example, a suitable service might be available, but it would be advertised over a discovery protocol the frame does not know, or it would be of a type not recognized by the frame. In such a situation, the frame should veto its own publication. In Continuum, we see this situations addressed in an evolving fashion. One option is to upgrade the frame code to a more complex frame that also knows about the additional service discovery protocol or the additional service type. A simpler option is to extend the frame-code repository at the middleware with an

additional frame that implements the same operation, but exclusively with services of the new type or advertised over the new protocol.

4.2.4. Vetoing the frame-publication decision

To help the user devise and carry out his activities related to the data viewed on the personal device, the relevant frames need to be published. The relevance is subjective, and depends on the user's situation, intentions, and environment, as he sees it. Unfortunately, the middleware can only partly infer these. We have conceptualized the difference through the notions of *a priori context* and *electronic context*. Figure 4.3 specialises Figure 3.5 and illustrates how the frame publication process takes place and what elements of the *a priori* context can influence it. The relevance and the semantics of *a priori* context elements are tightly linked to the particular objectives of a frame. For instance, if a user is a visitor in an office building (this is indicated by a special visitor attribute in the form of a role assertion, a notion introduced in Section 5.1) and by the location attribute of the device, then, say, a *DepartmentalPrinting* frame will not be relevant, but a *PrintInTheRoom* frame may well be.

A developer writing a frame could list conditions on the *a priori* context that warrant frame publication, including the type of data with which that frame operates. (An example of such a description is provided in Chapter 6.) This works for middleware that has simple publication heuristics and simple frames, but will not work in general. We provide evidence why some frames might need a customized frame-publication decision algorithm.

First, establishing the relevance of a frame to a situation might require certain elements of the *a priori* context that are not taken into account by the default decision algorithms. The frame might have to obtain those elements directly from sensors and other sources of *a priori* context. Second, most frames will not be relevant for publication if certain services are not available or are not accessible to a particular user.

Querying service directories can be expensive, so in some cases, one might want to opt for publication without first looking for services, which, in turn, can result in a large number of published frames. Solving

this trade-off is a per-frame decision. This further supports a design that gives frames the opportunity to make the decision themselves.

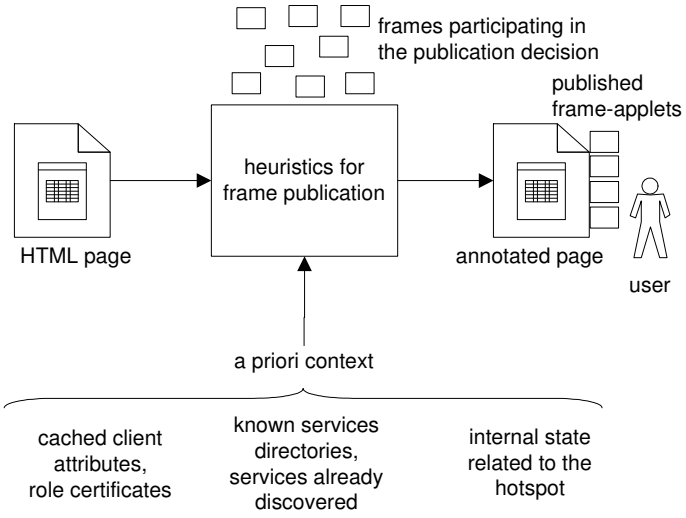

Figure 4.3. Frame publication

It is important to note that stale frames can result regardless of who takes the publication decision. Staleness can be due to user mobility or to services terminating or failing. This means that services that were discovered at publication time and had warranted the publication of a frame become inaccessible. A few service-discovery protocols provide their own mechanisms to deal with service termination or unavailability. Usually this means the use of leases, periodic advertisements of service presence, expiration dates for discovery, client notifications, or service-state introspection. As frames are aware of the particular discovery protocol(s), they can deal with these mechanisms.

Staleness can also result from optimistically publishing frames that later fail to find some of the services, or when services discovered are not accessible to the particular user.

In Continuum, both the notion that the published set of frames is an approximation and the notion that some of the published frames could become stale are exposed to the user. She can always opt to force a regeneration of the toolbox (i.e., rerun the publication algorithm for that toolbox). Given the dynamism, and the manifest link between service availability and user location, mobile users will have fewer expectations of service availability than stationary users.

4.3. Frame design issues

Having described the functionality needed in a frame, we note that we have refined the notion by implementing several frames for the Continuum prototype. In the prototype, applets are a combination of HTML and JavaScript, to capitalize on the functionality already available in web browsers. Applets are mostly responsible for the user interface of the frame. The frame servlets are implemented in Java.

Both assume a certain runtime environment provided by Continuum. The frame applets run within the web page to which they are attached, or in the webpage that refers to or embeds the objects to which they are attached. They rely on API support (HTML and JavaScript) for access to attributes stored at the browser or at the home node, for displaying their UI, and for standard communication with servlets over HTTP so that the middleware can intercept attributes and instantiate servlets. The developer can also rely on a library of UI templates that can speed up frame applet design.

The servlets run at the Continuum server. They can rely on Java API support for running frame-specified discovery queries over select discovery protocols, for transcoding between well-known data types, and for accessing attributes cached at the middleware, or stored at the browser or the home node.

Given the considerations of the frame functionality and the split applet-servlet design, a servlet needs two entry points. The execution entry point is called when the servlet is instantiated on demand, following communication from the frame applet. The input of this method is state information, the data object(s) on which the frame operates, and other information about the hotspot to which the frame is attached. The publication entry point is called when the frame has to decide on its own publication, accepting or vetoing it. The method takes as input information about the hotspot in conjunction with which the frame is consulted about publication.

This brings us to the life cycle of a frame as an applet servlet combination, which consists of one publication and one or more executions.

At publication:

1. from the frame code repository, the frame publication engine chooses a candidate frame for publication;
2. the corresponding servlet is instantiated to be consulted on publication;
3. the publication engine calls frame's publication entry point to veto or agree with the decision;
4. if the frame should be published the middleware instantiate the frame applet code and adds it to the web page;
5. the servlet is discarded.

At execution:

6. the user chooses to interact with the frame applet in the toolbox in the web page;
7. when and if needed, the applet initiates communication with the servlet over a standardised API;
8. this communication is intercepted by the middleware, which instantiates the corresponding servlet;
9. the middleware calls the execution entry point on the servlet;
10. the servlet is discarded.

Chapter 6 provides more details on how frames are implemented in the proof-of-concept implementation, including details about the APIs on which applets and servlets can rely in the prototype.

In the current design, frames are to be implemented as an applet and servlet pair. Each frame is essentially a proxy between the mobile and one or more services. While support for alternative designs requires changes of the Continuum server and its various mechanisms, conceptually, the design of frame can draw from the extensive research of proxying issues that has been done in mobile computing. Without intending to be a full taxonomy, Figure 4.4 illustrates the span of the proxy-design space, as reflected in the literature. The "wireless device" is used to mean the software on user's personal device, e.g., the web browser.

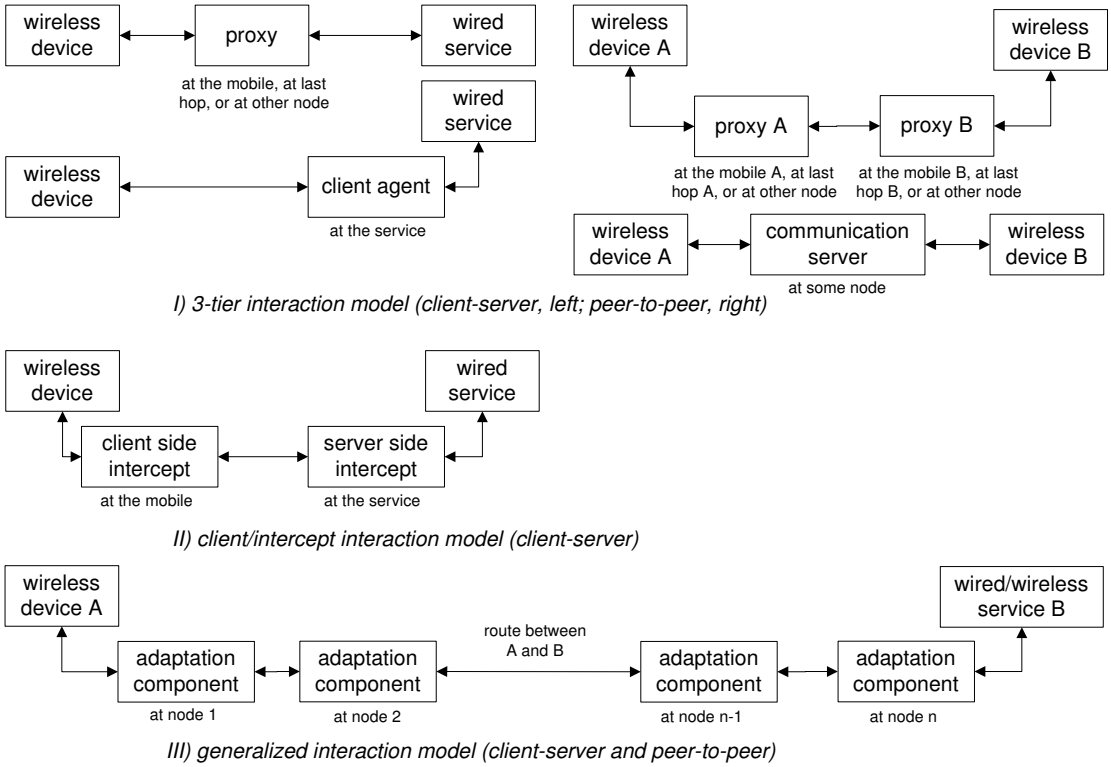

Figure 4.4. Proxy-based mediation in mobile interactions

In the three-tier model (Figure 4.4 I), the software proxy component interposed between the communicating peers can execute at the mobile, at the last hop (owned by the ISP or the middleware provider), or even at the service. Variations of the three-tier model have also been proposed, for instance splitting an HTTP proxy into two [45] (Figure 4.4 II). This is, essentially, how frame applets and frame servlets function. Proxying can be generalised to multiple proxies run at several nodes in the path between the communicating nodes (Figure 4.4 III). This is often used in streaming media projects [140, 13]).

Peer-to-peer interaction refers to the interaction between two wireless devices, for instance, a PDA that controls a wireless digital camera directly, or downloads pictures from it. This can happen through two proxies that communicate directly over wireless, or they might communicate through some communication server (like the Continuum server).

4.3.1. Frame publication mechanism

Returning to the current design for frames, the description of a frame life cycle made reference to two important elements of the design of the Continuum middleware: the frame publication engine and the repository of frame code.

The frame publication engine synthesises the electronic context. The better it can approximate the service area of the services the frame needs, the better the publication decision. Determining when a client is in the service area of a service is in general an open problem, as service areas can be defined in arbitrary attribute spaces. Approximating it can be considerably easier. Any component of the *a priori* context can also influence the frame publication decision, as can the frame properties specified by the developer. In this thesis, we seek to create a general framework that allows different heuristics to be employed to approximate the ideal publication decisions.

Because, in chaotic ubiquitous computing, many communication networks can be available simultaneously, the Continuum server might not necessarily be physically close to or even aware of the user's location. Still, it can be aware of some of the device attributes. In addition, Continuum devises a framework that combines the acquisition of credentials with the acquisition of bootstrapping points for service discovery, and involves users in these acquisitions. This framework is described in the next chapter. Here we note that it enables the publication engine to compile a list of service directories that can use to obtain references to services relevant to each individual user. They can be used to discover services that in turn can be used to determine which frames should be published. Closer coupling between Continuum and a physical location may provide an even better set of frames to access local services.

The current prototype implements a simplistic algorithm. The main limitation is that only the data MIME-type and the type of hotspot are used in the publication decision. Considering information that is more complex is delegated to the frame. For each hotspot in a document, and for each frame that refers to the particular type of hotspot, the publication engine calls the publication entry point for the frame. Essentially, frames can veto their publication at this point, or they can perform particular initialisation

steps. The frame-publication engine has access to meta-information about each frame to be published: the servlet main class, the files with templates to generate applet code, and the hotspot type. The servlet also uses this information for its publication decision, as well as for other functionality, such as building discovery queries. More details are provided in Chapter 6.

Devising sophisticated heuristics for frame publication is outside the scope of this thesis. We note that heuristics for frame publication have to deal with two scalability aspects. From the system's viewpoint, the publication process needs to finish in a reasonable amount of time. From the user's viewpoint, the number of published frames must be limited to a number one can cope with.

In the proof-of-concept prototype, there is no limit on the time a frame can spend to decide on publication, and this can be a problem. The time it takes to publish the frames directly affects the latency perceived by the user, from the moment a web page is requested until she can interact with the tools added to the page. To a certain extent, workarounds can be found. The toolbox could be published in a separate HTML IFRAME, so the latency penalty is paid only when the user interacts with the toolbox. By that time, the publication process might have finished. Several candidate servlets could decide concurrently on their publication, and the corresponding applets could be sent to the browser as they become available. A carefully designed web browser will display this frame and allow interaction even if the entire toolbox is not available yet (essentially, this translates to the browser rendering pages even if they are incomplete).

Nevertheless, the fundamental problem remains: a faulty, malicious or simply poorly coded frame can interfere with the responsiveness of the ubiquitous computing middleware. Consequently, sandboxing techniques similar to the ones used for mobile agents could be used.

In spite of all these technical issues, our proposed approach opens a fertile research topic on how to improve the ubiquitous computing experience by devising innovative frame-publication heuristics, either system-wide or within a frame.

Tighter coupling between the middleware provider, the SPs covering a certain area, and the frame developers might allow the publication decision algorithm to capitalize on knowing, for instance, that a

user is a visitor or an employee. (The acquisition of such roles is discussed in the next chapter.) Specialised frames might use any elements of the *a priori* context in their publication decision, e.g., video cameras and image recognition.

To avoid overwhelming the user with tools, global heuristics could track which frames are used most often, and increase their likelihood of publication. This presumes more evolved heuristics to select frames, such as a greedy algorithm. One could also track the history of aggregated frame usage from all users, to infer popular activities in a certain area, and hence favour certain frames.

In this thesis, we do not deal with the potential for phishing and other spam-like abuse related to frame publication. Limiting the set of frames per provider and/or enforcing signed frames for non-repudiation, combined with a mechanism to blacklist abusive providers, can alleviate such problems.

As previous chapters show, the user is aware that the set of published frames can only be an approximation. Ideally, she should be given means to contribute proactively to the extension of this set, usually through special frames the middleware provides. Simple and not necessarily user-friendly examples of such frames could ask for additional URLs to "local" service directories, or allow the user to register code for new frames.

To get the most up-to-date electronic context, technical solutions to refresh the set of published frames can be imagined, such as moving all the frame-applet JavaScript and HTML code to a special document (say, referred to from an HTML IFRAME), with zero validity period and automatic refresh. We did not prototype this because, conceptually, it brings nothing new. Indeed, given the dynamism of the environment, a frame can become stale even while the user interacts with it.

4.3.2. Dissemination of frame code

A ubiquitous-computing middleware provider is expected to install, configure, and maintain the Continuum server running the middleware, as well as the frame code base. We see the set of frames at the Continuum server as expanding dynamically and possibly on demand, to suit the preferences of the middleware provider's users and the mix of services available in the area it covers. Given these

considerations, frame code management becomes important, as does the way in which Continuum comes in the possession of frame descriptions and code. Devising sophisticated mechanisms for frame-code dissemination is outside the scope of this thesis. However, we do enumerate several possible approaches.

The middleware providers may implement a number of frames that use standard service types and they may commission frame code from independent software vendors. The licensing expenses can be recovered from the flat access rate they can charge (authenticated) users.

A middleware provider will probably also take a lead role in negotiating business agreements with third party SPs in its service area. If the middleware providers are also ISPs, they might include frame-code exchange and sharing in their roaming agreements with other ISPs offering ubiquitous-computing middleware. Such agreements would ensure a similar level of service in a foreign network.

Conceptually, one could also imagine services that push frames to user devices, which in turn share them with their Continuum server, to be reused for other mobile devices in the area. Of course, such frames have to be signed digitally to prove they indeed come from a particular SP.

Assuming inexpensive local storage, many frame applets could be cached at the mobile or pre-fetched, to save on transfer latency. However, the publication decision is still better taken at a well-connected network node (i.e., the Continuum server).

In theory, the SPs themselves could offer the code directly to user devices, but it is likely that a third-party level of indirection (the frame) can better take into account user preference (e.g., localization), foster competition, and combine services from different providers. In addition, this simplifies the infrastructure for small SPs, such as a corner store.

To summarise, essentially, we regard the notion of frame as cognitively related with the philosophy behind UNIX commands. They are very specialized, with a data-centric, minimalist, software interface with the ubiquitous computing infrastructure, yet, with arbitrarily complex functionality and user interfaces. Chapter 2 contrasts the Continuum frame with related software abstractions, either from ubiquitous computing or from mobile and distributed computing.

Our approach of defining a small software unit that interfaces directly with the user as the building block for ubiquitous computing is confirmed by the observation of Newman *et al.* [83] that ubiquitous computing systems should support the user in assembling available resources to accomplish her tasks, since it is virtually impossible to have applications for each task needed. Moreover, Erickson [31] notes that humans should be left in the loop because of the discrepancy between what the user expects and what the context-aware system performs. The next chapter describes a framework for service discovery and authorization that starts from the notion of role assertions and involves users in acquiring them.

Chapter 5

A role-based discovery and authorisation framework

The previous chapters introduce the notion of making ubiquitous computing functionality orthogonal to other web-application logic, and show how, under these conditions, middleware can mediate interactions with contextual services. A suitable software abstraction and its functional properties are devised. After an introductory discussion of the layers of complexity in the ubiquitous computing paradigm, this chapter introduces the premises for the discovery and authorisation framework in Continuum. We then describe a mechanism that combines service discovery with the acquisition of role-based access credentials (called *role assertions*). We show how the combination can be exploited easily and effectively by users, and support this with a description of the prototype implementation for role-assertion exchange. This is followed by a discussion of the way this design facilitates flexible authorisation and access control performed directly by the target service, without special-purpose client devices or software. The chapter closes with a discussion of the issues associated with role assertions, and the potential for future work related to the discovery and authorisation framework, followed by a short summary of how the home node supports this framework and other functions of the middleware.

Mediation in ubiquitous computing is hard due to the inherent complexity. Figure 5.1 illustrates the multiple layers of complexity with which such a middleware has to deal (note that in Figure 5.1, the layers are vertically aligned). The electronic interactions are constrained by the physical layout (I). Short distances in network or data-link-layer terms do not always match this physical reality. The need for physical access (e.g., the key from a room), can further restrict the interactions that are possible. Overlapping administrative domains (II) might or might not influence what services the user chooses, but they do affect what technologies are deployed and they dictate usage restrictions. The network access technologies available in a given area and on particular devices shape the nature of the ad-hoc interactions

(III). Various services could interact with either personal or public devices (IV), but the extent of these interactions is constrained by the realities of the other layers. In particular, non-physical/non-technical constraints such as the role of a user (V) or device attributes can have an impact on what interactions can take place. To complicate things even further, in addition to interacting over the internet, devices might also interact directly, peer-to-peer (VI), over network technologies that preclude middleware involvement (e.g., infrared, Bluetooth).

The Continuum middleware incorporates these layers of complexity into its assumptions. Middleware for ubiquitous computing has to span mechanisms that bring all these layers together. In particular, this chapter presents contributions in linking roles (V) and service discovery (IV), and exploiting peer-to-peer interactions among user devices (VI).

The Continuum framework for service discovery and authorisation starts from three observations:

- users will want to access services outside the administrative domain, or the smart-space domain as configured by its designers;
- discovering the topology, in terms of services are available there, is fundamentally linked to the acquisition of credentials to access the services; and
- a mobile user in an unfamiliar environment is aware she will probably need to take an active part in seeking credentials and discovering what she can do in that environment.

The main difficulty is that, at design time, the set of users, and the kind of services and service providers that might interest them are not known. In an airport, a resident of city where the airport is, a tourist in the city, a commuter to the city from a different city, or a business traveler, might each be interested in different contextual services, particularly if we are referring to services accessible from the airport but not necessarily related to flying.

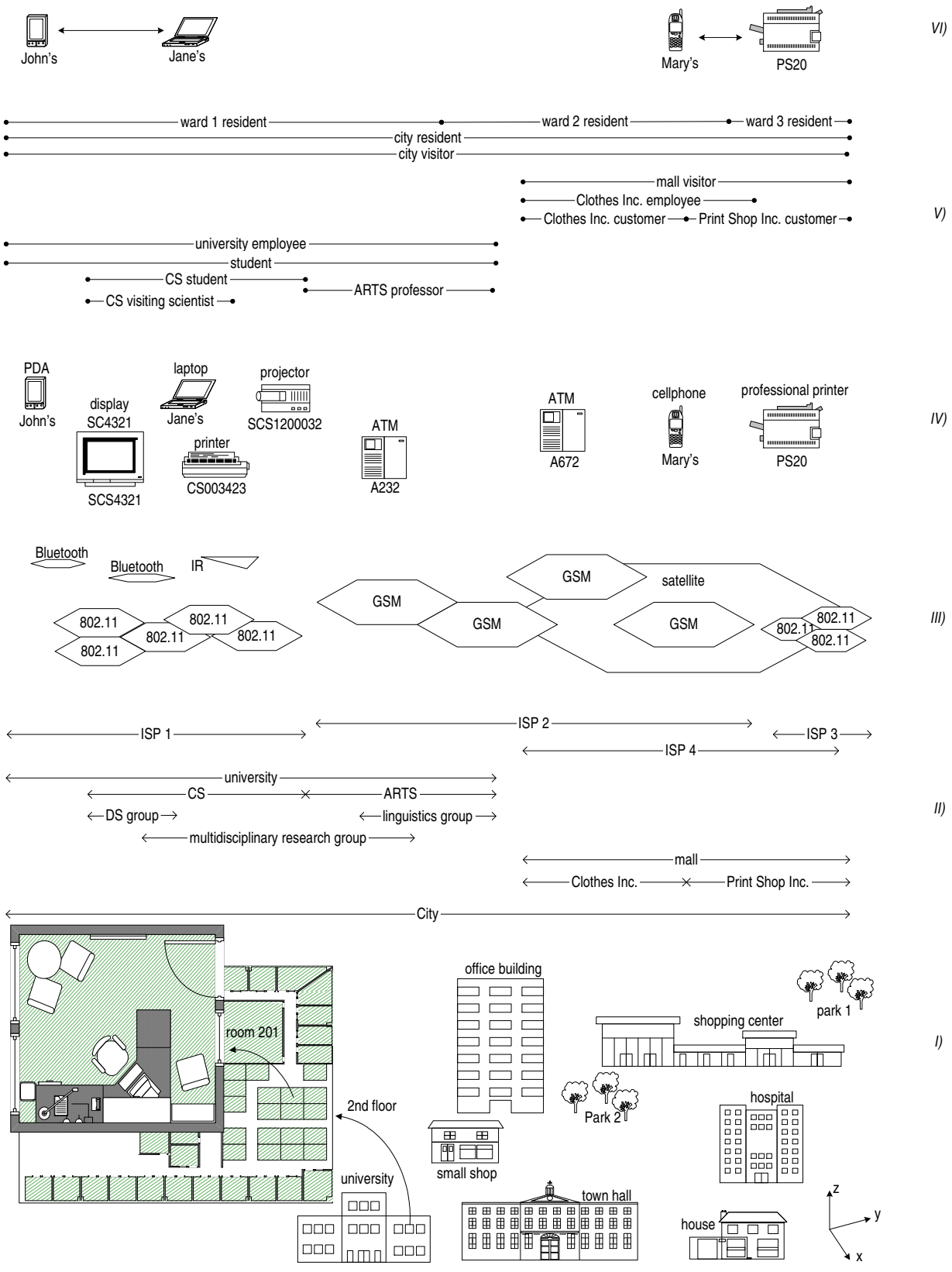

Figure 5.1. Complexity layers that middleware in ubiquitous computing has to consider

In Chapter 2 we discuss the various context elements considered in the literature and survey several mechanisms to capture the *a priori* context. In particular, we discuss the state of the art in service discovery in ubiquitous computing. A clear conclusion to that discussion is that, on their own, none of the mechanisms mentioned there is flexible enough to accommodate multiple administrative domains. Moreover, different domains might opt for arbitrary combinations of technologies to provide their services, so yet another service-discovery standard would provide little value.

In the next section, we introduce a flexible mechanism for discovery of relevant service directories, independent of the service-discovery protocol. We envision that middleware for ubiquitous computing will in fact use a variety of mechanisms to find out about the service directories that may contain services relevant to each user. This accommodates the decentralization specific to the ubiquitous computing paradigm.

The same realistic provisioning for decentralisation dictates our assumption that the access authorisation is at the discretion of each service and its owner or service provider (SP). Services, in most cases, do not have prior knowledge of the client, in the sense that an SP cannot know in advance the identities of all the users that will ever be its customers. Hence, access control lists (ACLs) of user identities are not directly useful as a basis for access policies. Sometimes, the proof of device location can be used for authorisation. In general, service-access eligibility comes from a certain quality of the user with respect to a situation, an SP, or a third party. For example, the fact that a user is a guest in the hotel authorises him or her to use a printer in the lobby. At the time of printing, physical presence in the hotel/lobby might or might not be required. One can imagine that during the normal check-in interaction between the user and the hotel clerk, the user's device receives some token that attests to the user's role as a guest. This means that from a technical point of view, the device needs to acquire a piece of information that becomes a dynamic attribute (property) of that device. Later, that attribute can enable access to consenting services.

In other words, service access means not only locating the services but also acquiring the attributes needed for access authorisation. In most cases, the user will have to acquire these credentials actively, and this is another opportunity for ubiquitous-computing middleware to help the user. One of the contributions of this thesis is devising such a framework for active acquisition of credentials, starting from the observation that this acquisition can and should be linked to service discovery.

5.1. Role assertions

Our approach is centred on the concept of a "role assertion" that says the holder is acting in a particular role for select SPs. Middleware at the edge of the network can broker the acquisition of signed assertions, effectively contributing to enabling ubiquitous computing. Solving the challenge of service and role discovery with multiple administrative domains involves two key concepts. A particular role assertion must be created and communicated to an interested user, and then used to secure access to a service (potentially in combination with other attributes). Section 2.6.2 gives an overview of work related to access authorisation and role assertions.

To make the middleware involvement in service discovery, authentication, and authorisation possible, there are several prerequisites:

- suitably populated service-directories,
- provisioning for the generation of role-assertions,
- deployment of services that use roles as a basis for authorisation, and
- code dissemination for frames that facilitate access to these services.

An SP must arrange for its services to be listed in one or more service directories, and the underlying technology for those services is at the choice of the SP. As we shall discuss later, the assertions may list the URLs of several service directories that accept the role. The roles and their semantics are also conventions established by the services accepting them, hence by the SP.

The generation and acquisition of roles is user driven. The issuer and signer of the assertions can be the SP itself or any business partner (or select user) whose signature is acceptable. The role assertions are acquired through interactions mediated by tools published by the Continuum server. This assumes both interacting parties use the same Continuum server. Alternatively, an assertion could be acquired independently through direct interaction with a signing authority other than another user.

The user interacts with services through the mediation of the frames. During its execution, the frame invokes the external service or services, supplying acceptable role assertions. The service may engage in some type of validation of the assertion, and it might request additional attributes.

The SP or the middleware provider has to ensure that the frame-code repository contains frames that exploit the services the user encounters. The SP could ensure that directly, by writing or commissioning frame code and then disseminating it. Alternatively, it can just adhere to standards. For instance, a printer following a standard protocol and advertised with a standard discovery protocol can be accessed through a general-purpose printing frame that knows the protocols.

Figure 5.2.Role assertion example: A hotel guest

Figure 5.2 is an example of an assertion issued and signed by a hotel clerk, confirming that a certain user is a guest in the hotel. This assertion is acquired during check-in. The role is granted to a pseudonym. At authorisation time, it is expected that the service will verify that the client presenting the assertion belongs to a user with that pseudonym. The assertion also points to service directories that list services which use the role as part of their authorisation algorithms (the *Service Directories* field).

Before discussing in more detail who issues roles, how are they exchanged, how authorisation is performed, and the implications for frame publication, let us show how combining a middleware proxy with the role-assertion framework can support different sample scenarios.

In a first scenario, a small hotel provides a rich ubiquitous-computing environment to its guests. The human users involved are a prospective guest with a web client on her mobile device, and the registration clerk. The administrative entities are the hotel company and the ISP, and they are assumed to be distinct in this scenario. The ISP also acts as a Continuum middleware provider. The servers involved are the user's home node, the Continuum server, and possibly the back-end server of the hotel, acting as home node for the clerk. Examples of local services are a printer for guests in the hotel lobby, a vending machine, a cash dispenser, etc. An incoming guest first needs to establish a session with Continuum through some use of the network. This results in the initial publication of tools relevant in the local context. The guest may or may not have business relationships with individual SPs or the specific ISP. As the guest checks in, she and the registration clerk arrange for an electronic role assertion to be acquired by the browser on the guest's PDA. More precisely, the role-assertion attribute will be stored by a frame applet as part of a cookie, and may be stored at the user's home node.

The role assertion of Figure 5.2 gives two service directories where the guest can find services accepting the role: an SLP directory agent maintained by the hotel, and a Jini service locator maintained by a local third party. Later, Continuum can detect the role assertions in outbound applet requests, and use the role and directory information to publish frames for services that accept the role. In this way, those services have no need to create or maintain a relationship with the guest. When the guest tries to access one of the services, the service has simply to accept the role granted by the hotel employee (clerk123@HotelName.com in Figure 5.2). A service may also apply whatever checks or validations it wishes to verify the authenticity of the user. More details on the protocol exchanges among the hotel clerk, the guest, the proxy, and services follow.

Since it wishes to create and sign its own role assertions, the hotel must negotiate role names and their semantics with other local SPs. If it also wishes to make services available to holders of its roles, the hotel needs to advertise those services in its own service directory, or in some public service directory maintained, say, by the ISP. If some of its services are non-standard, the hotel may have to develop custom frames and disseminate the code to ISPs covering its area.

As a second scenario, consider a guest visiting a university. Also, assume authorised faculty members can create role assertions for their academic visitors. These assertions could be presented to get access to the services installed and maintained by the department, and even to get access to the university network. When the network itself cannot be used to transfer the role assertion from the professor to the visitor (the public is not given access to the university network), peer-to-peer communication between the professor's PDA and the visitor's could be used. The professor interacts with a Continuum frame to create the role assertion. This is transferred to the visitor's PDA. This can be over, say, Bluetooth, or even through a USB flash drive, handed to the visitor. Such peer-to-peer exchanges assume the visitor's PDA knows what to do with the assertion so that it becomes visible to frames accompanying web objects.

In a slightly different scenario, several ISPs might cover a shopping mall area, without necessarily being affiliated with the mall. The various services provided by the mall merchants are registered in service directories maintained by the ISPs, the mall management, or by other parties. In such a scenario, visitors have a trust relation with one or more of the ISPs covering the mall (and probably the city). To benefit from these trust relationships, mall merchants could agree to accept role assertions signed by the ISP. The ISP might base these assertions on its technical information about the user's location.

These simple scenarios illustrate the variety of situations the framework can accommodate. Having shown how role assertions can enable a variety of scenarios, we provide details of the implementation of this mechanism. The user is likely to visit unfamiliar locations with unknown services available (familiar environments are an easy special case). Because the user is in an unfamiliar location or situation, she is aware of having to participate actively in discovering and exploiting the electronic context, as well as

acquiring and exploiting various local credentials for use with local services. This is similar to asking someone for directions, acquiring a room key in a hotel, using a conference badge to gain access to an event, or arranging payment for a service by presenting a credit card.

Such credentials (role assertions) could be issued by a service of the SP, of authorised ISPs, or other SPs. However, while technically issuing role assertions can be automated, delegation decisions made by human users cannot be always anticipated or coded in policies. Our framework effectively allows users to "gossip" with other users about credentials and other local information, independently of SPs and ISPs. Examples are users exchanging role assertions in electronic mail, or beaming them between handhelds, storing them on some medium for use with a different personal device, and storing them in persistent storage provided by a service trusted by the user (such as the user's personal computer). For services that need only trivial authentication, these assertions can be used without involving the user in intrusive verification steps.

Continuum enables delegation of access rights in a manner similar to how users interact in everyday situations. For example, the person that has the key to a certain room can let a guest into the room without further approval or involvement from a third party, and without requiring the person to yield the room keys (although this can happen too, for short periods of time, if the recipient is deemed trustworthy by the holder of the keys). Without changing the semantics of who can use the printer, the usage can be made much more convenient. The host Alice can make Bob a "guest printer-user" which would allow him to print temporarily. In the absence of such a mechanism based on role assertions, Alice, who is allowed to print, can take the more troublesome route of getting the document file from Bob, printing it, and then handing the printout back to Bob. While this would still allow Bob to print, this rather cumbersome procedure has to be repeated for each document and each of Alice's visitors.

The emulation of a "word of mouth" process in exchanging role assertions tries to capitalize on the local information that more knowledgeable users in an area have already discovered, or that local users already possess. A local user is a user that knows more about the topology and the authorisation

requirements of the services of a certain SP or group of SPs. Such a local user might also have the authority to delegate or give access rights. With the same goal of improving the user experience, this exchange of information adopts from the opposite perspective to initiatives such as WS-Federation [135] and Liberty Alliance [64, 65]. Their notion of federated identities (the global set of attributes for aggregated user accounts and the handles to access them) standardises how the federated websites share user identities and attributes.

In Continuum, users exchange role assertions by interacting with a special-purpose frame. This frame for issuing or acquiring a role assertion appears in Continuum Tools, a set of frames published by default, which contains services that do not take user data objects as input. In this chapter, we keep the description of the exchanges at a design level. However, for clarity, we will refer screens captured from the corresponding frames in the proof-of-concept prototype. The details of this prototype are discussed in the next chapter.

Figure 5.3 shows the details of the process. *AcquireRA* (interaction 2) and *IssueRA* (interaction 4) are the two frames published for the recipient and issuer, respectively. The interactions labelled (5) correspond to a three-way handshake in which the recipient chooses to acquire a role after verbal interaction with the issuer, so that Continuum correctly associates the users' sessions. Then, the role assertion is generated (marked with an R in Figure 5.3) and returned to the recipient. Continuum retains a cached copy of the role assertion to aid publication heuristics. The assertion can be stored at the client's home node. This makes sense when the recipient uses multiple personal devices that need the role. Alternatively, the main copy of the assertion can be stored only at the device. The home node of the issuer is involved in executing cryptographic computations to create the assertion. When it is not accessible, the issuing device can do the computations itself, provided reasonable capabilities and it has the secrets needed to generate and sign the assertion.

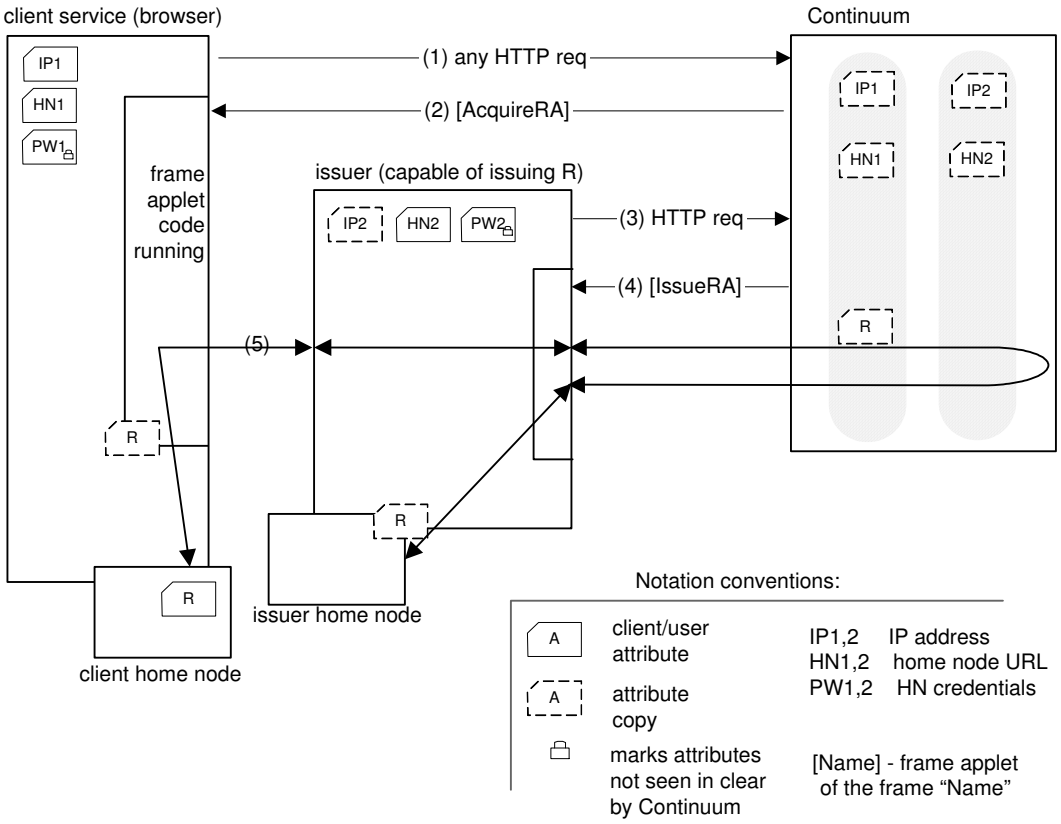

Figure 5.3. How a role assertion is generated and acquired

Figure 5.4 combines the successive screens seen by the guest and the clerk from the hotel example, as they interact to issue the role assertion of Figure 5.2. The numbers beside each image indicate the sequence in which the two users' actions occur. The upper part of the figure refers to the issuer, the bottom to the recipient. They begin by choosing *ContinuumTools*, and then either the *AcquireRA* or *IssueRA* frames (in the figure, the user interface of each frame uses the label "Acquire Role" and "Issue Role," respectively).

We assume each participant chooses a simple password, then, in (2)-(5), each participant can enter his own password and the password communicated verbally by the peer. In this way, the middleware avoids mediating interactions between two unless they are face-to-face and have agreed to engage in the exchange of role assertions. Users' privacy is protected by ensuring that the presence of the user in a certain area and the user's identity are not divulged to anyone else without the user's consent.

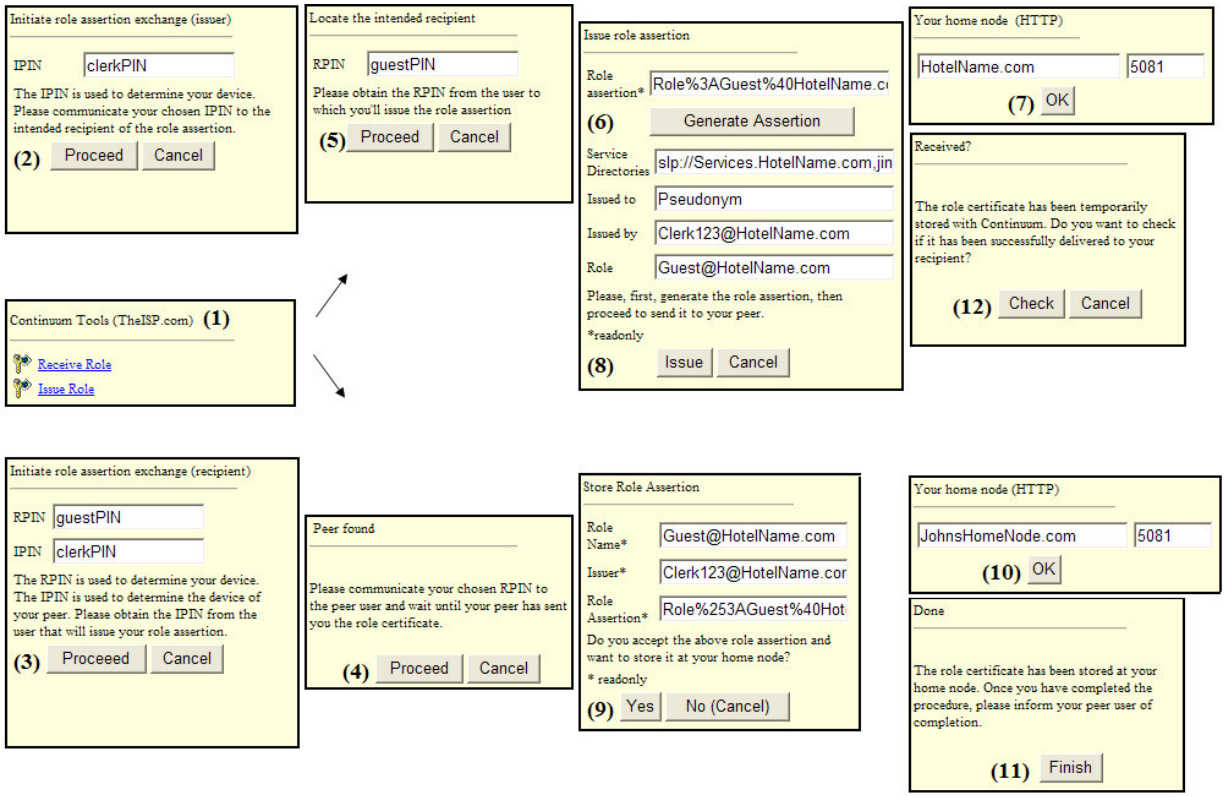

Figure 5.4. How the user acquires a role assertion

After the passwords are exchanged and the two sessions are paired, in (6)-(8), the *IssueRA* frame uses the clerk's home node to generate the role assertion, assuming this is too onerous to accomplish on the clerk's PDA. In step (7), the clerk confirms the home node URL obtained from the attribute pool at the browser. This can be followed immediately by a challenge for credentials, coming from the home node. Then, the assertion is received and stored at the guest's home node in (9)-(10), possibly after a challenge for credentials. The procedure ends after confirmations in (11)-(12).

Figure 5.3 and Figure 5.4 assume the two peers use the same Continuum server. The example can be extended to using different cooperating servers that trust each other to have the same privacy and security standards for handling user information.

Having a mechanism to exchange role assertions requires SPs to decide what role assertions their services will accept. This includes what roles are acceptable and who/what can delegate them. The set of services that need to understand the structure and semantics of a role assertion is limited to the issuing

service and the services that accept it for authorisation. For the Continuum server, most fields of the role assertions, including the role name, can be treated as opaque information.

5.2. Service-driven access authorisation

Assuming a frame has been published, the user has chosen to interact with it, the frame knows a suitable discovery protocol and has discovered a service with which it knows how to interact, then the service can to be accessed and the access needs to be authorised.

In the hotel scenario, suppose the user wishes to print a data object on the printer in the hotel lobby, and that the printer is owned and operated by a local copy center. Assume an arrangement between the hotel and the copy centre, which stipulates hotel guests may use the printer without hindrance, upon successful presentation of the assertion for the Guest role. Whether the copy center charges the hotel for guest usage and whether the charge appears on the guest's bill is a separate issue, left to the discretion of the hotel and the copy center.

Figure 5.5 describes the general steps for using a service in this manner. The user chooses to interact with the Print frame in (3). For generality, *ServiceOp* is used in Figure 5.5 as a label. The service was discovered before the frame was published or is being actively discovered (not shown) when the user initiates interaction with the frame, by interrogating the directories listed in the role assertion. For the hotel example, that might have yielded all the printers in the hotel, and the user might have chosen the one in the lobby. The applet provides to the servlet the URL of the main copy of the data for which the frame was published.

The frame servlet requests access to the service in (4), and is subsequently challenged for attributes, in particular, for an appropriate role assertion, the Guest role assertion is this case. The service may determine that further attributes or validation are required, such as some assurance that the end user is indeed in the possession of the private key corresponding to the pseudonym of the role-assertion holder in steps (7-10). Once authorisation is obtained (11), the service is invoked with the input data provided in (3). The data referenced in (3) is converted to a format required by the service in (12). Because at the

browser, only an altered (transcoded) version might be available and this is not what the frame semantics say will be printed, the main copy referenced in (3) is converted to Postscript and sent to the printer in the lobby. All the exchanges (3-14) are part of the execution of the frame.

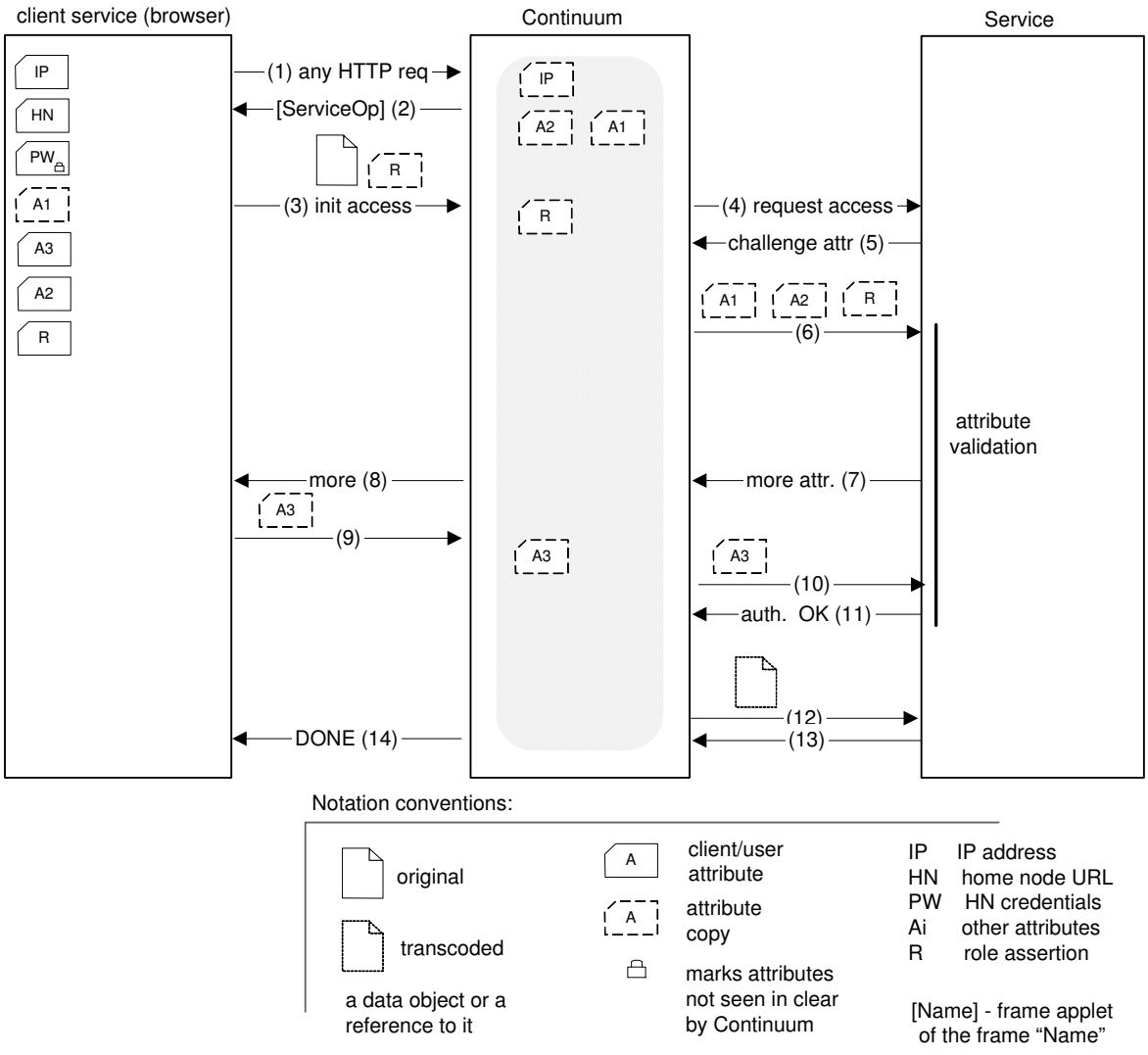

Figure 5.5. Service-driven access authorisation

In general, the authorisation decision is based on the attributes of the user requesting access. Authorisation policies may be arbitrary complex. For example, location-based authorisation might not be enough. Some services might authorise on a combination of location and certain role assertions. Other services might accept user-password credentials for local users and role certificates for guest users. Some services might require proof that the user is indeed aware of the interaction, as opposed to the interaction

being carried out by the frame. How this can be proven is beyond our scope. Authorisation might be mutual: user personal devices could also authenticate the service.

The middleware, through frames, can be involved in the authorisation process. In Chapter 3, we discuss how a soft session at the Continuum server caches attribute values. The sessions are instantiated when the particular user device starts using a middleware server. This might just mean creating a blank session for the user device (i.e., with no attributes cached initially). Subsequently, it can be populated and updated as the middleware monitors further communication between the frame applets and servlets, and possibly further interactions by the frames with external services.

Alternatively, the user might login to the middleware. Figure 5.6 shows the details of such a login procedure with the middleware provider. The middleware intercepts and responds to any HTTP request by publishing a tool (the *Presentation* tool) that requests information about the user. In a trivial implementation, this can be a simple HTTP form. In a more sophisticated one, the frame can obtain attributes through introspective JavaScript code. The response (3) establishes a cache of client attributes within a session, attributes that later are used in the publication heuristics. After session establishment (4), the middleware would normally transmit the reply to the original HTTP request (1).

Logging into the middleware provider might or might not be simultaneous with getting network connectivity. Personal devices can obtain network-layer connectivity in a technology-dependent way. The user normally has a prior business relationship with one or more (wireless) ISPs. In many current wireless networks, a login procedure is required for the mobile to gain network access. The login can be user-assisted or it can be performed automatically by the device. Likely, this will also be the case in the future. For some technologies, the identity of the user is known. For instance, a GSM network verifies the account information on the SIM (Subscriber Identity Module) card in the phone. In other cases, like authenticated WiFi LANs, the user will need to be queried. Such WiFi networks in hotels and universities, including the University of Waterloo, often resort to HTTP-based authentication. The user accesses a

well-known URL where he enters credentials. Prior to this authentication, any other traffic is blocked by the access point.

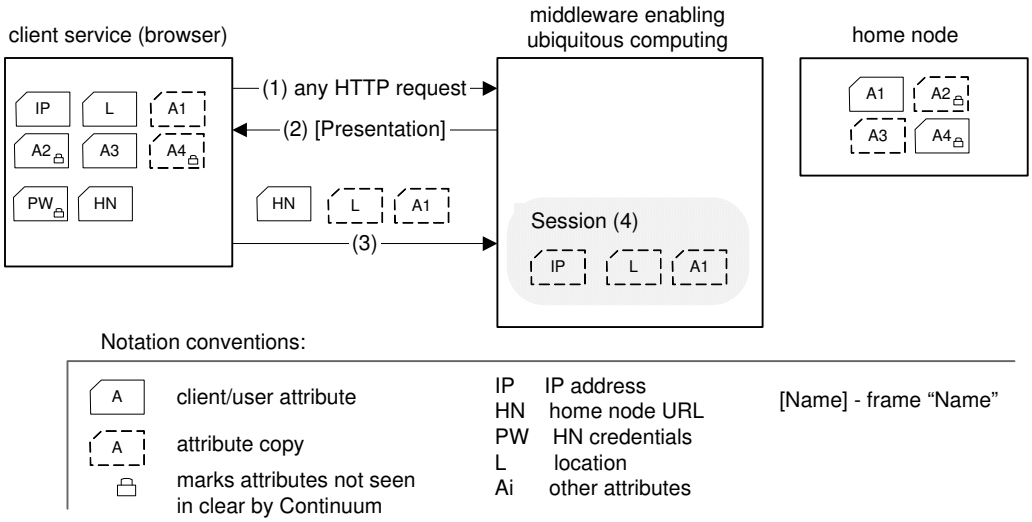

Figure 5.6. Establishing a session with the middleware

Ubiquitous-computing middleware might be offered and maintained by the ISP, running at the last wired hop. When the user logs in to the ISP's network, he implicitly signs up with the middleware. Alternatively, the middleware can be maintained by a third party. The user might not actually trust, say, a free access point. In such a case, the communication with the middleware has to be encrypted end-to-end.

Returning to how the attributes in the cache at the Continuum server are used, we note that issuing a role assertion does not mean the user delegates all her rights. The exact semantics of this partial trust transitivity depend on the policies of the services accepting the role. Holding the same role certificate can result in different access privileges, depending on how the target service uses the assertion.

Authorisation routinely includes validation of the credentials presented by the user, as can be seen in Figure 5.3. Validation means verifying that an attribute provided by a device is indeed an attribute of that device. The validation procedure depends on the attribute and on the service doing the authorisation.

Consider the example of location. Its validation is a hard problem because there are many possible access technologies, location technologies, and ways to represent location (e.g., room, floor, building, or

x, y, z coordinates). Ubiquitous-computing middleware should be flexible and not commit to a single location technology. For example, WiFi access points might provide coarse location information as a signed, time-stamped assertion. If the service trusts the ISP to provide only the true location of subscribers, validating such location statements means validating the signature and the signing key with a trusted certification authority or a chain of certification authorities ending with a trusted one. GSM networks usually cooperate with the device to determine location information. If the client is not trusted, when the location is determined autonomously by the client, additional validation might be required, possibly in the form of a third-party confirmation. Location validation techniques can go as far as using special-purpose hardware that can verify proximity [11]. Location validation is a complex problem that is beyond the scope of this thesis.

As with location, verifying user identity depends on how the identity is expressed. It could mean proving the device knows a private key, providing a password, performing a biometric scan, etc. In the validation process, the service might contact third-party servers, a situation not explicitly shown in Figure 5.3. For instance, Pluggable Authentication Modules (PAMs [106]) in UNIX and DCE allow different methods of verifying identity (password, Kerberos, smartcard, reading the thumbprint, eye iris, etc.). Each method can have trust coefficients, which can be combined for more reliable authentication. This was considered in several ubiquitous computing projects, such as Gaia [80] and Aura [122].

To validate a role assertion, the service verifies the signature and then validates additional attributes such as the identity or pseudo identity-included in the certificate. Specifying identity is a private matter between the issuing service, the service that later accepts the role assertion, and the role exchange frame. Further verification with the issuer or with a third-party repository of trust-delegation operations could be needed, depending on how the role assertion defines validity.

In the current prototype, the identity of the issuer (*Clerk123@HotelName.com* in Figure 5.2) and the identity of the recipient (*Pseudonym* in Figure 5.2) are arbitrary strings entered manually. Realistically, these strings should be a public key (for the clerk) and either a public key or a pseudo-key (in the sense of

a short-lived public key or a symmetric key) for the recipient device/user. Validating role assertions is linked to identity management. The way to specify identity is largely an open problem that has not been completely solved by the current PKI (Public Key Infrastructure) standards [36]. However, the issue is not the focus of this thesis. Existing protocols for verifying that the holder of a ticket is the original recipient (e.g., Kerberos [82]) cannot be used directly in this case. The issuing service is not an authentication service. It cannot store all the keys, for all the services that will ever accept a *Guest* certificate (for example), because this set of services can change arbitrarily.

Continuum imposes no particular view on how services evaluate or validate attributes. Some services, for example, might accept a role assertion without any interactive verification. This is not unlike the off-line credit card transactions, where the card is accepted and the potential frauds are dealt with later, outside of the transaction context, through, say, legal action against fraudulent use or insurance against fraud. During the validation, it is possible that the service will request additional attributes needed to validate the initial set. In other words, the validation of attributes is a recursive process.

Since authorisation is at the discretion of services, legacy services need special support. So do other services that do not implement authorisation and attribute validation themselves, as a way to simplify the implementation or the management. There are several solutions. First, a frame could implement custom verification. This has the disadvantage that the SP has to trust the middleware with authorisation decisions. Alternatively, the frame could perform authentication and authorisation negotiation with a special-purpose service. As a third possibility, the SP may just essentially proxy its pool of services. The authorisation would take place out-of-band with respect to the protocol for accessing the service (relying on a challenge-response mechanism with the user to validate attributes). For instance, in the case of a standard LPR printer, authorisation has to be done outside the LPR/LPD protocol. Lower-level mechanisms could prevent unauthorised access to the service (e.g., IP packets could be dropped at the network level). This amounts to having in place a distributed firewall. The out-of-band authorisation

assumes there is a way to identify the packets of the service-access requests of an authorised client. Unfortunately, TCP, IP, or data link filtering cannot protect against IP and/or MAC address spoofing.

5.3. Discussion on the uses and issues of roles and role assertions

Esentially, the framework presented in this chapter links service discovery and role-based access control, with the assumption that users are aware they are "strangers" in certain areas, and that they need to acquire certain credentials (role assertions). This is much like asking for directions when in a new place. Role assertions help the ubiquitous-computing middleware bring to the user the services she might need to access, even if those services belong to different administrative domains. The framework requires no single point of authority and preserves the loose coupling and the decentralisation specific to the ubiquitous-computing paradigm.

Hence, a mobile device will accumulate a set of role assertions, related to the current activities of the user, because users are actively involved in the assertion acquisition, and they accept them explicitly. Consequently, the service-directory URLs the middleware collects represent a tailored set of discovery bootstrapping points that improve the chances that the set of published frames matches the user's interests.

Gaining knowledge of discovery bootstrap points from role assertions can be combined with other mechanisms. As outlined in Chapter 2, some service discovery protocols, notably Jini [120], have mechanisms of their own to discover service directories, usually in the same physical network. One could also leverage research results in direct discovery of web presence for physical services from the CoolTown project [60]. A business relationship between select SPs and ISPs may also allow for a mechanism that lets the formers register service directories with the ISPs covering their area of service, much as local businesses ensure their advertisements are printed in the local newspapers.

Without enough ubiquitous-computing systems deployed in real conditions, it cannot be inferred today how real users would use the paradigm and how real SPs would choose to deploy their services, so one cannot accurately evaluate with how many role assertions a user will typically have. If this number proves to be large, one might need mechanisms that partly automate the acceptance of role assertions by

mobile devices, provided such mechanisms can be created. Having too many role assertions can also have an adverse impact on the publication engine, because it might have to query too many service directories in the publication process. This might be possible to alleviate by assigning an adjustable relevance coefficient to each role assertion, so only the service directories from the most relevant assertions are used.

A user might be in the situation of acquiring two role assertions for the same role. The simple way to deal with this is to overwrite the older role assertion, with the user's consent, even if it was issued by a different issuer. Complex schemes can also be imagined. If both assertions were kept, the union of the service directories would be used for discovery purposes. For authorisation purposes, one of the assertions could be used first, and if the access failed, it could be retried with the second assertion, and so on.

An open issue is how users know they can issue a certain role assertion, and how they learn the exact semantics and implications of issuing those assertions. In the current design, it is assumed users are instructed in that respect outside the technical framework. The SPs that would accept the role assertions issued by certain users would also have to inform those authoritative users of their abilities.

As with most of ubiquitous computing (see Chapter 2), there are potential security threats to which the framework could be susceptible. The issuer of a role assertion might be a malicious user, trying to hijack a role assertion exchange and replace the assertion with a fabricated one. She could do this just to interfere with the user experience or to make the user access certain services. End-to-end encryption or validation of the role assertion when the user accepts it can partly alleviate this. When the malicious user is an authenticated user, the situation can be dealt with outside the model. If a user steals or fabricates a certificate, and the frame engine uses roles to determine what frames to publish, at most, such a user will force some additional frames to be published, and that only in the case the publication engine does not validate frames. She will not be able to use those frames, because the services they access would reject accesses accompanied by a fake role assertion.

A more delicate issue is how to avoid giving the middleware access to certain credentials and attributes needed to access services and the home node. Conceptually, such credentials would be sent encrypted or would be sent over encrypted channels (SSL). However, this is a technical issue. The frame applet has to be a combination of JavaScript and HTML, and current browsers lack user-interface mechanisms to convey easily to the user which server is the target of an HTTP request. The browser address bar does not show the queries issued by JavaScript or HTML, and the information in the browser status-bar is quickly overwritten. Consequently, the user cannot reliably know when she is interacting directly and in cipher text with a service, or when the interaction passes in the clear through Continuum.

The dynamic code of the Continuum frames is not necessarily trusted for all purposes. Short of pages obtained over SSL, there is no way to enforce such separation between credentials being handled by the browser itself, credentials being handled by JavaScript in the browser, and credentials being handled by a potentially malicious party such as an HTTP proxy. It is also hard or impossible for the user to distinguish between security domains. For example, when the user is a guest and is challenged to prove his identity to a printing service, he might not be able to establish easily if the printer indeed belongs to the organisation to which he is a guest.

Clearly, browsers need user-interface support to indicate conspicuously what is going on, and runtime support for access to attributes. A first small step in current browsers is the use of a visually different dialog box for password challenges, indicating the server. This can, however be, faked by the proxy if no encryption is used. These browser issues and other user-interface issues are outside the scope of this research. Currently we assume the user trusts the middleware provider to maintain the confidentiality of his data.

As mentioned previously, we envision the framework will be used in situations where a breach in security, regardless of its nature, is only a nuisance that can be dealt with outside the model through traceability, non-repudiation, and law enforcement. When this is not the case, virtual private networks and end-to-end encryption are the well known technologies that should be used for role assertions.

Returning to uses of role assertions to improve the user experiences, we note that devising suitable roles is a hard design issue. An obvious example is to match them with smart spaces. A role could also capture the notion that the user is involved in a certain activity. Other examples from the literature have been reviewed in Chapter 2.

Depending on what roles represent, they could also be used to decide which frames to publish. If a role represents an activity, frames suitable for that activity might be published. This would imply a closer coupling between the Continuum middleware and a physical location or select local SPs. The middleware or certain frames in the code repository would have to understand the semantics of select roles. This means that somehow SPs, which decide role semantics, would have to disseminate that code to the middleware. This essentially implies more or less conventional business agreements to maintain Continuum in a locale.

Another future work area related to role assertions is what other information can be included in a role assertion to help the middleware improve the user experience further. A role assertion might describe the physical area in which the role is valid, similar to the idea of a service area [70], to help the publication process. It could also describe what additional attributes are normally needed for role validation. This means including information on the conditional validity of the role. Such conditions can be enforced by the issuer (an example is the expiry date in Figure 5.2), or by the target service. Since the issuer has more knowledge about the "local" setup in the area where the role will be valid, a role assertion might also include a description of a legitimate service, in the opinion of the issuer. This could be expressed in terms of attributes of a trustable service in the view of the issuing service user, like the public keys or the X.500 certificates proving a service belongs to a certain company. The frame would have to validate such attributes presented by prospective services before they are accessed.

To end the description of the framework for discovery and authorisation in Continuum, we restate that the role assertions contain both credentials the recipient can use to access services in a different administrative domain, and bootstrapping points to start the discovery of services in those administrative

domains. Roles and role assertions are not mutually exclusive, so, the user chooses what hat she is wearing. She does that initially by accepting a role assertion, and subsequently by choosing to interact with a particular tool.

5.4. The functionality of the home node

To round out the design of the Continuum architecture, before presenting the proof-of-concept prototype in Chapter 6, we briefly summarise how the home node supports the framework described in this chapter and the other functions of the middleware. The notion of home node was introduced in Chapter 3. It is run by the user's desktop computer, or it could be purchased as a service from a third party. It can offer persistent data storage, store and serve user attributes and preferences, and participate in specific computations. In the following, we discuss a number of possible features of the home node.

The user might want to persistently store a certain data objects she currently browses with her mobile device. Since it can get lost or destroyed easily, the mobile device is not a very reliable storage place, but the user could store such the data object at the home node.

Certain attributes might not be available at the client (where the frame applet can access them), or might not yet be cached in the Continuum session (where the frame servlet can access them). In addition, updates to the applet or servlet copies may be needed to avoid staleness. There is a trade-off between refreshing the attribute periodically, and accessing it on demand. This is due to the high latency of access, as several roundtrips are required to get an attribute, due to technical peculiarities related to JavaScript access across pages coming from different servers. A per-attribute trade-off decision is likely.

When the user can have multiple devices, it makes more sense to store a copy of assertions at the home node, if there is a chance that they will be used from different mobiles of the same user. When the role assertion is issued to a pseudonym, the home node also has to store whatever information is needed to make the pseudonym known to all the devices belonging to the same user.

When the mobile is too poorly equipped to handle the computations to generate the role assertion, the home node can also assist in the computation. This is a less and less likely situation with the new

improvements in hardware. Other functions are also possible. For example, the home node could maintain a profile with user trust and privacy preferences, and some automatic authorisation could be possible. On the other hand, it has been argued that "automatic trust" cannot exist, and consequently "automatic security must be fictional too" [36].

The home node could also provide a runtime for the execution of belated operations. We define these to be normal operations in whose finalization the user cannot participate. For instance, when the user is trying to send a document to a copy store, but she cannot remain connected to ensure that the printing finishes without error. The mobile device would just get a confirmation that the belated operation has been initiated by the home node and the Continuum server. Subsequently the user or one of her devices could get a confirmation that the job has finished. Users find such confirmations useful. As a small study with the Satchel system [62] has discovered, many users complained that, when faxing a document, confirmations were only that the document has been queued instead of being about whether the fax has been received at the recipient. As a natural extension of the notion of belated operation, the Continuum home node could also become a runtime platform for agents acting on behalf of the user.

This thesis does not investigate further the usefulness of having such functionality implemented at the home node. So far, we have presented a design that that shows how middleware can enable ubiquitous computing at the edge of the network. The next chapter describe a partial implementation of this design in a proof-of-concept implementation. The exercise of building this prototype incrementally has helped in refining the core notions of the Continuum architecture.

Chapter 6

The Continuum prototype

This thesis shows how middleware at the edge of the network can efficiently enable ubiquitous computing, including how the electronic context can be presented to the user in a manner orthogonal to web applications, and what software abstraction and discovery and authorisation framework support this. In this chapter, we describe Continuum, a proof-of-concept prototype implemented to validate and refine our ideas. A general description of the user experiences with Continuum is followed by an account of the technologies and components used, and the assumptions about how to configure and deploy Continuum. We then describe how the server functions internally, how a Continuum frame is developed, and what runtime support is offered to the developer. The chapter ends with a summary of limitations and future prototype extensions.

Figure 6.1 shows how Continuum is experienced by a laptop user of Internet Explorer (IE). Reasonable care has been taken when coding the frames and the Continuum runtime for applets to ensure that the altered pages function correctly on several mainstream browsers, including Netscape for Windows on laptops, Konqueror and Opera for Sharp Zaurus PDAs, and PocketIE for PocketPC PDAs. However, lacking tools to capture PDA screens, we show here only IE screenshots.

The upper left panel of Figure 6.1 shows a typical web page, rendered in the user's browser, without Continuum. In the upper right panel, the same page is shown with Continuum, before the user clicked on any hotspot to see the associated toolbox. The images in the page (the title artwork, the title GIF, and the photo in the article) each have an associated toolbox that can be accessed by clicking on the corresponding image. The entire web page also has a toolbox, which can be accessed by clicking on the hand icon in the upper left corner of the page.

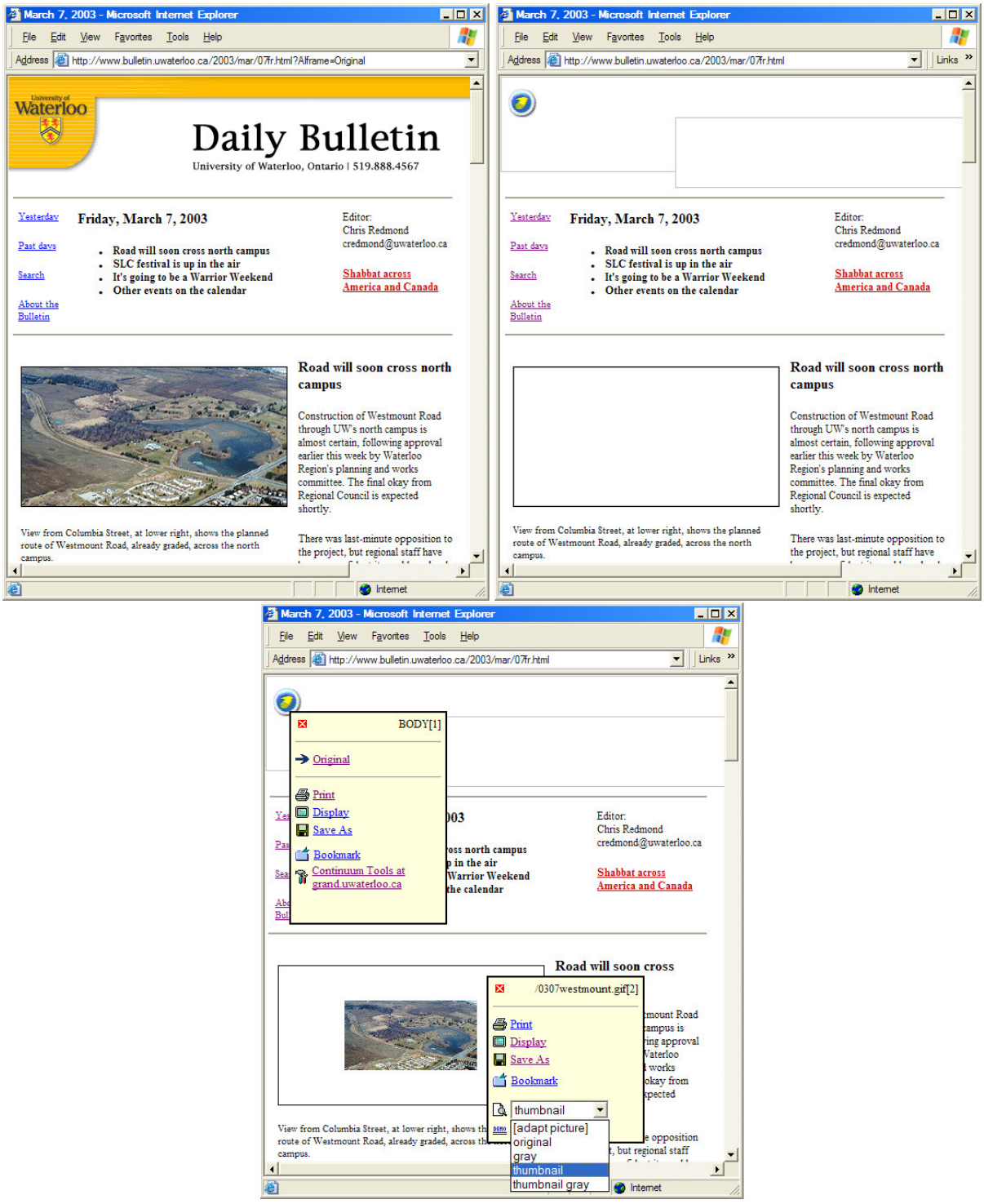

Figure 6.1. Continuum, as experienced from a laptop browser

The bottom panel of Figure 6.1 shows the toolbox for the document and the one for the photo. The user has already invoked the last tool in the image toolbox menu to display a thumbnail version of the

image. By default, in the Continuum prototype, all images are replaced with blank GIFs, a procedure that normally results in smaller file sizes. Ideally, all images should be replaced with the smallest blank square on which the user can click. However, we have found that browsers consider the *height* and the *width* of an *IMG* tag as read-only attributes. When an HTML document provides no values for them, they are automatically initialised to the dimensions of the first image to which the *src* attribute of the tag points. Later, when the browser is instructed to change the image displayed for that particular tag (by changing the URL to which *src* points), these values are not updated. The *IMG* tags without *height* and *width* could be fixed when Continuum adds hotspots to the page, but this would require downloading all the images before the page could be returned to the browser, which introduces an unacceptable delay.

6.1. Technologies and components

Having described the general experience of the user interacting with Continuum, we provide details of the technologies and components that make up the Continuum prototype. From an HTTP perspective, the Continuum server runs as an HTTP [95] proxy. Most web browsers can be configured to find their proxies automatically through schemes such as the web proxy auto discovery protocol [35]. The proxy configuration can also be manual. The user device is assumed to run a standard web browser.

Recall the elements of the Continuum architecture in Figure 3.3 (page 51): the user's wireless devices, the web servers, services of arbitrary service types and access protocols, directory servers for arbitrary discovery protocols, the user's home node, and the Continuum server mediating interactions among them.

In the prototype, it is assumed the user device runs a standard HTTP browser with support for cookies and JavaScript. The user model is defined in more detail in Chapter 1. The web servers serve standard HTML. These pages might contain their own JavaScript. While the JavaScript and HTML of the frame applets can potentially interfere with the original JavaScript in the page, we found this a rare occurrence on the admittedly small set of pages tested. However, since this interference is a real possibility, we expect a production version of the prototype would use a different scheme to insert frame applets, like

grouping them in a separate document (in an IFRAME) to force their execution in a different JavaScript context.

The SPs can choose any access protocol for their services. This is possible because the service-access protocol is private to each individual frame. We have prototyped frames that access printers over the line printer daemon protocol (LPD [94]), displays over the X protocol, and the home node (itself a service) is accessed over WebDAV [99], an extension of HTTP. The SPs also have the choice of what service discovery protocols they use to advertise their services, and in which service directories they advertise them. Currently, we support SLP directory agents [100] and Jini service locators [120]. The simplifying assumptions about the attributes of the services and especially about the location are described later in this chapter.

The user's home node is prototyped as a WebDAV server, accessed over the standard WebDAV protocol [99] and relying on standard HTTP authorisation mechanisms. Only basic HTTP authentication is currently supported. Basic authentication does not use encryption. Using encryption, by changing the authentication scheme or by using one SSL connection from the browser to Continuum and one from Continuum to the home node, does not pose any conceptual difficulties. The harder problem is to make sure the Continuum server does not have access to the home-node credentials. As discussed in Chapter 5, most browsers do support SSL, but there is no obvious indication in their user interfaces to convey to the user which secure web site the JavaScript code is contacting. Hence, since the interactions with the home node happen from within a frame, and the frame-applet JavaScript code is added to the page by the Continuum server, short of the user reading the page code, there is no easy way to verify that Continuum does not gain access to the credentials. We do not attempt to solve this user-interface problem, and as previously mentioned, the prototype assumes the user trusts the Continuum server (usually run by her ISP) to not misuse credentials.

The home-node implementation was only complete enough to test our ideas about frames, and so is rather rudimentary. In addition to the authentication shortcomings mentioned above, the home node does

not yet have the ability to execute code. Where this ability was needed, the code was included within the servlet, with plans to move the code to a module running at the home node. For instance, the generation of role assertions in the *ExchangeRoleAssertions* frame should happen at the home node, to avoid the middleware gaining access to the private key of the issuing user, but currently, this happens in the frame servlet. The JavaScript code in the frame applet is not a good place for such computations either, because it is foreign code coming from Continuum. The home node does not use SSL either, and this would have not solved the problem of conveying to the user the identity of the web server to which the browser communicates.

The home node stores all data objects sent to it in a directory hierarchy. A special directory stores the attributes as individual text files, with all non-printable characters escaped. Currently, the only attributes that can be store are role assertions. The credentials to access the attributes at the home node are the same as the credentials needed to access any other data stored there.

6.2. How Continuum operates

Figure 6.2 gives a bird's-eye view of how the Continuum server operates. Like all web-based servers, once Continuum intercepts a request from the client browser, the request is run through a dispatch module that directs it to the appropriate component. This is the point where the incoming request is matched with a particular session from the session pool. If such a session does not exist, it is created. The Continuum sessions are soft, in the sense that the information there can be regenerated if lost or expired. This session information consists of cached client attributes and known URLs of service directories relevant to the user. (Several service directories available to all sessions are also maintained.) We imagine that a cache of relevant services, either as a system-wide pool or in each session, would be useful in reducing latency.

Client attributes are cached and refreshed as Continuum intercepts standardized applet-servlet communication. The cache is initiated when the first request from a particular client reaches Continuum. Currently, the prototype does not include a login procedure (for how this would work, see Figure 5.6) nor Continuum-server hand-off, since sessions are soft anyway.

In the general case, a good choice of session identifiers depends on the network access technology and the topology of the network. One requirement is that session identifiers should facilitate Continuum-server handoffs within the same middleware provider or to different providers. Potential candidates are the IP address, the MAC address (for WiFi), the subscriber identity (for GSM), or some other unique user identity. User identity, instead of device identity, enables multiple devices of the same user to share the same session, allowing for the development of more interesting frames.

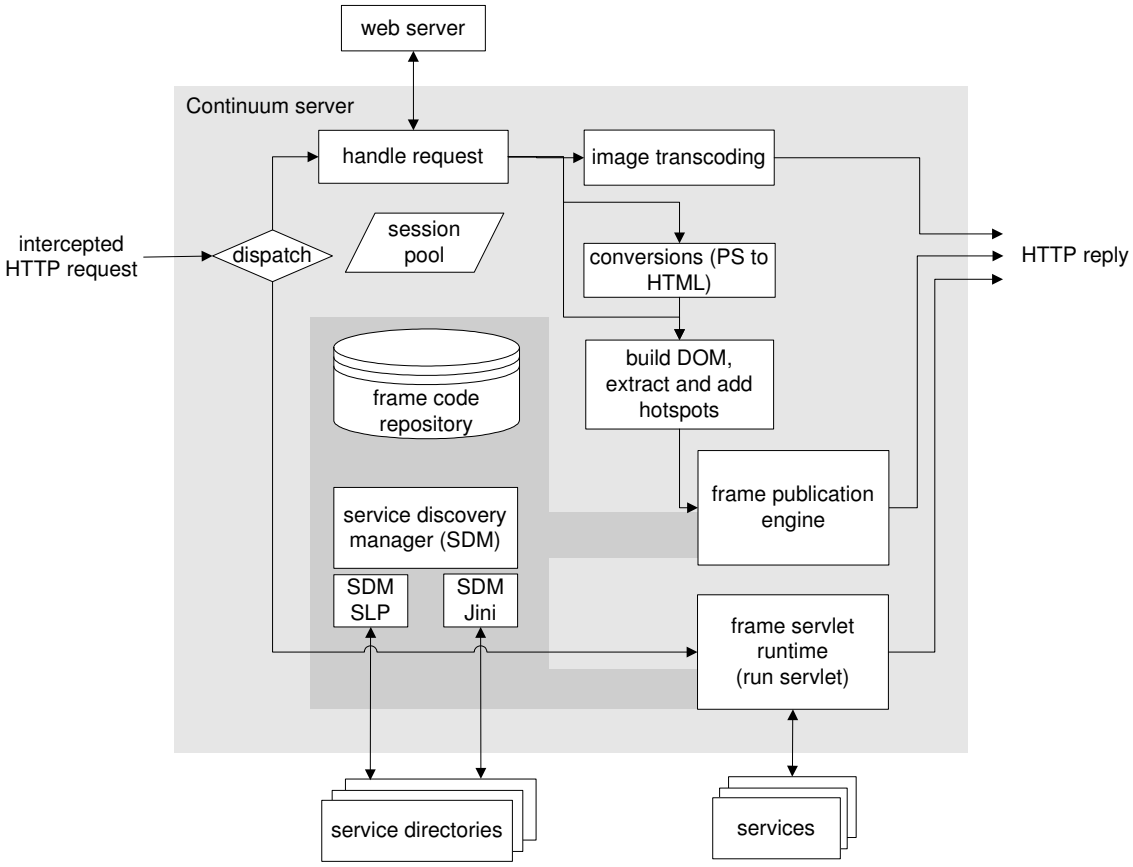

Figure 6.2. Continuum server internal flow

In the current prototype, the session ID is a concatenation of the Continuum-server IP address and the IP address of the client. The first part avoids confounding new clients that have published frames left over from previous Continuum servers in a chain of handoffs, frames that might contain references to the session content on the server that published them. The client IP address is unique within the server

network. The assumption that only one user can be associated with a device is not a big limitation, because the client devices we consider are personal devices without concurrent users.

Returning to the dispatch module in Figure 6.2, if the HTTP request is a normal request for a document, the appropriate web server is contacted and the document is retrieved. At this step, since the Continuum server deals with multiple clients, a site-wide document cache can be used. This well-understood technique could be easily integrated with Continuum.

Once the document has been retrieved, it is transcoded, depending on its type. When an image is requested, a blank image of the same size is returned, unless, as discussed, the URL is a surrogate that identifies the request as made by the *AdaptImage* frame applet, in which case the appropriate transcoding operations are applied to the image before returning it.

When an HTML document is requested, the document object model (DOM) is built, and the hotspots are identified or inserted into the document. When the requested document is a Postscript file, the document is first transcoded to an HTML document that embeds a gif image of the first page, under the default assumption that the target device cannot display the format. The original mime-type information is preserved, so at publication-time it is known that that the hotspot is associated with a surrogate HTML document created from a Postscript document. Consequently, special frames for navigating the document are published. Navigating the document means changing which page is displayed as a GIF.

The types of hotspots that were prototyped are Continuum-tools, HTML document, link, picture, and transcoded-Postscript. Figure 6.3 shows sample toolboxes for each of them, from left to right. The Continuum Tools hotspot is not associated with a particular document, and can be accessed from all toolboxes, regardless of the hotspot with which they are associated. The HTML-document hotspot is coupled with the *BODY* tag in the document. The link hotspot is coupled with an anchor tag (*A*). The picture hotspot is coupled with an *IMG* tag. The transcoded Postscript hotspot is coupled with the *BODY* tag of the surrogate HTML document that replaces a Postscript file.

The toolboxes, as holders for the user interface exposed by each published frame, are a combination of HTML and JavaScript. The HTML mainly draws the toolbox, while the JavaScript code displays and hides it. The toolbox also offers an applet-execution runtime consisting of JavaScript methods that applets can expect to be defined. We return to this topic later.

When all the hotspots and the code for toolboxes have been added to the DOM, the latter is taken as input by the frame publication engine, together with information about the initial request and other information cached within the session. We have already discussed in Chapter 4 how the frame publication occurs. The publication engine uses a frame-code repository. The servlet for each candidate frame is instantiated and consulted on the publication decision by calling the *onPublication()* method of the servlet.

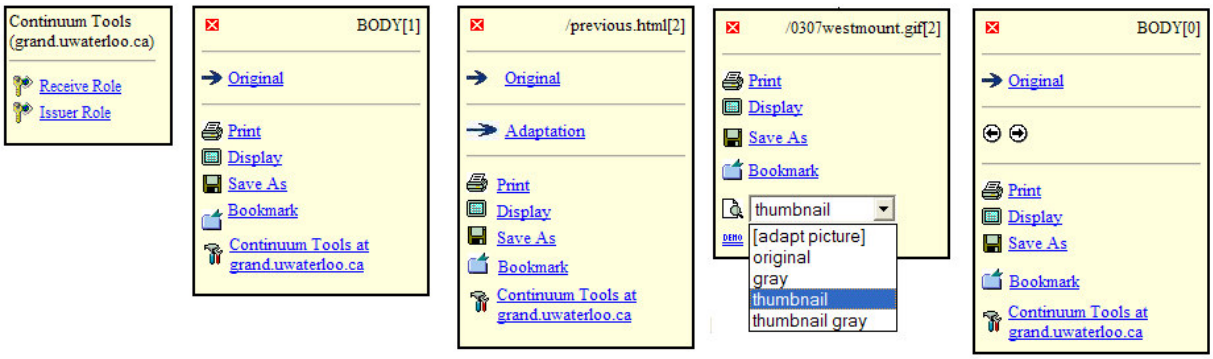

Figure 6.3. Examples of toolboxes for several hotspot types

Having a frame-code repository is essential to the extensibility of the Continuum server. Rudiments of mechanisms are in place to manipulate the repository dynamically, but a better understanding of the dynamics of the relation between the SP and ISP is needed before more specialised management can be prototyped. We leave the study of this open problem to future work. We envision that ISPs and SPs will have business agreements to ensure smooth operation, and that different ISPs will have roaming agreements in place. These agreements will also specify frame-code sharing and dissemination.

In the prototype, each frame-code bundle is characterised by a meta-information record that directs how the frame is instantiated. The next section contains more details and examples on this topic. In short, this record specifies what HTML and JavaScript template files are used to generate the user interface for

the frame applet, the servlet class, the kind of hotspot to which the frame applies, and client attributes for which the publication engine will generate code for automatic collection. The frame can collect other client attributes and input parameters on its own, through its custom interface.

After the publication engine has annotated the DOM with code for displaying toolboxes, the applet runtime code, and the dynamically generated code for the published applets, the corresponding HTML document is regenerated and returned to the browser. As discussed in the previous chapters, the user can continue normal browsing, that is, navigating hypertext or interacting with the web-based application to which the web pages belong. In addition, she can also interact with the published frames.

Returning to Figure 6.2, the last possibility is a request for a servlet. The developer of a frame has to follow a standardised method for interaction between the applet and the servlet. The latter uses name-value pairs as part of a web query that often is attached to URL requests for the web object to which the frame refers. When relevant, the request also specified the session.

An example is the *AdaptImage* applet that generates a request like the one in Figure 6.4 (I) to ask for a thumbnail as shown in Figure 6.1. To force the browser to issue the request, and display the result as a new image, the applet modifies the *src* attribute of the *IMG* tag accordingly.

As another example, when the user clicks the *Find Service*s button of the *Display* frame in Figure 4.1, the applet generates a request like the one in Figure 6.4 (II). The *discovery* command indicates to the servlet what to do when the *onInvocation()* method of the servlet is called. The servlet subsequently uses the Service Discovery Manager built into Continuum to run the frame-generated discovery query for displays. The location, specified by *AIbuilding*, *AIfloor*, and *AIroom* attributes, is used as appropriate for each discovery protocol. Since there is no standard SLP template defined, only displays advertised with Jini are supported. After the target displays have been found, the servlet returns to the applet an HTML document containing information identifying each of them.

Once the user chooses a particular display from the list, the applet generates a zoom request handled by the *onInvocation()*method of the *display* servlet. The request looks like in Figure 6.4 (III). (The

developer must follow the conventions at least for the way the servlet is invoked.) The Jini object that proxies the display is specified by its class (*AljavaClass*), its user-friendly name (*AIname*), the URL of its service locator (*AIsdsurl*) and its unique service identifier within that service locator (*AIsrvidh*, *AIsrvidl*). Note how in each case the session is specified when irrelevant.

I) GET **http://www.bulletin.uwaterloo.ca/images/2003/0307westmount.gif**
?AIframe=**AdaptImage**&AIsize=**small**&AIcolor=**same**

II) GET /?AIframe=**Display**&AIbuilding=**DavisCentre**&AIfloor=**3**&AIroom=**DC3552D**&
AIcommand=**discovery**&AIactonurl=**http%3A%2F%2Fwww.lib.uwaterloo.ca%2Fdiscipline%2F**
Cartography%2FImages%2Fkw.gif&AImimetype=**image/gif**&
AIsession=**grand.uwaterloo.ca-bowman.uwaterloo**

III) GET /?AIframe=**Display**&AIbuilding=**DavisCentre**&AIfloor=**3**&AIroom=**DC3552D**&
AIclientHostName=**bowman.uwaterloo.ca&**
AIcommand=**execute**&AIactonurl=**http%3A%2F%2Fwww.lib.uwaterloo.ca%2Fdiscipline%2F**
Cartography%2FImages%2Fkw.gif&AImimetype=**image%2Fgif**&
AIjavaClass=**adapt.jinisrv.JiniDisplayService&**
AIname=**boyne**&AIsrvidl=**7554844631297791501**&AIsrvidh=**2153157648221816062**&
AIsdurl=**jini%3A%2F%2Fgrand.uwaterloo.ca%3A4160%2F**&
AIsession=**grand.uwaterloo.ca-bowman.uwaterloo.ca**

Figure 6.4. Examples of standardized applet-servlet communication

For a frame, all the state is kept in the applet. This maintains the design requirement that the Continuum sessions be stateless. To save on the amount of information sent back and forth over the wireless link, frame designers can rely on the attributes cached in the session. Nonetheless, they must also provide fallback mechanisms, in case certain attribute values are no longer cached. A fallback mechanism can be as simple as returning an error to the user or asking her to retry.

Once the dispatching module has determined the appropriate servlet class, the servlet is instantiated. After pre-processing, the parameters of the request are fed to it. Normally, a request for a servlet is indicated by the presence of the *AIframe* parameter in the query.

When the servlet finishes acting on the request, it returns an HTTP reply to the browser. It is up to the applet to deal with the reply. For instance, the reply can also call a method of the applet to display a message to the user.

Both the publication engine and the servlet runtime make use of a Service Discovery Manager built into Continuum. Frames can interact with it to avoid implementing specific discovery protocols. However, they must still be aware of the semantics of the service-type hierarchy and of the service attributes for each protocol the frame knows about. The service-discovery protocols supported by the frame are a choice of the frame designer.

The Service Discovery Manager accepts the discovery query as an unstructured collection of name-value pairs. The client frame must use the correct names for service attributes so the service manager can build the discovery query correctly, according to each protocol. While a developer-friendly solution, this is a compromise between mapping between the attributes of different discovery protocols and defining a separate interface for each protocol. We also wanted to avoid defining yet another service-type hierarchy. The disadvantage of delaying error checking until the execution of the query is balanced by the ease of adding support for additional discovery protocols, with their own sets of attributes and service-type hierarchies.

In the current implementation, the service-discovery manager knows about SLP [100] and Jini [120]. To run queries, it uses fixed service-directory URLs and URLs acquired from the role assertions seen for the user of each session. The fixed URLs are discovered through broadcast when the Continuum server bootstraps.

In implementing the queries, several assumptions are made about how location information is provided. Location is considered at room-level granularity. In SLP, the notion of *scope* is used to group services. We assume that the scope (a string) equates to location, that is, all devices in a room belong to the same scope. When the default scope is used, the implementation assumes the location is irrelevant. Certain standard SLP templates explicitly specify the location as an attribute for that particular type of service. The printer SLP template has the *printer-location* attribute. In such cases, Continuum assumes the same value for both the scope and the location attribute. In Jini, the location of a service is usually an instance of *net.Jini.lookup.entry.Location*, a class that is part of the standard Jini package. Since we needed

consistency between the way the location is expressed for services advertised over SLP and services advertised over Jini, we made the convention that a Jini *Location* object with the respective values for the member fields *building*, *floor*, and *room* converts into the SLP location string "*building/floor/room*," and vice versa. These assumptions are by no means general. The appropriate way to represent the location and converting between different location-technology formats are problems beyond the scope of this thesis.

This section has presented how the Continuum server operates. The next discusses several examples of how to implement a Continuum frame.

6.3. How to write a Continuum frame

Writing a Continuum frame consists of implementing the applet code (that executes within a browser), and the servlet code (that executes at Continuum), and preparing a meta-information XML record that controls how the pieces of code are assembled and what other characteristics the frame has. Figure 6.5 gives an example of such a record, for the *Display* frame in Figure 4.1.

The applet and the servlet are instantiated independently in the life cycle of a frame. The meta-information record indicates the MIME type to which the frame applies, and if it applies to target objects embedded in the page (e.g., for a picture with an *IMG* tag) or to referenced objects (e.g., with the anchor tag *A*). The publication engine uses this information to decide if a frame should be published.

Once the engine decides the frame should be published, the servlet is instantiated. The frame name in the meta-information record also determines the Java class that is the entry point for the servlet code. Frame servlets implement a special interface that contains the *onPublication*() and *onInvocation*() methods, to be called at publication and at invocation respectively. As a matter of design, servlets are stateless so they are instantiated as needed. At frame-publication time, the servlet is instantiated, potentially to veto the decision taken by the publication engine.

If the servlet does not veto the publication, the applet code is assembled from the HTML and JavaScript templates specified in the meta-information record. Macro-substitution for well-known variables is performed. Such macro variables are the name of the current Continuum server, the name of

the toolbox, and the URL, the user-friendly name, and the MIME type of the data object to which the frame refers. For the record in Figure 6.5, there is no custom JavaScript code, only custom HTML code. As can be seen in Figure 6.6, this code is very simple for the *Display* frame. It just defines two table cells in the toolbox, containing an icon and a label, both calling the applet runtime function *gatherInputAndCallServlet*(). When the applet is instantiated, macro substitution is performed on tags such as *<AI:ADAPTIVE-INFRASTRUCTURE/>*, *<AI:MENU-NAME/>*, *<AI:URL/>*, and *<AI:MIMETYPE/>*. The *gatherInputAndCallServlet*() function invokes the servlet after it has gathered the attributes specified in the meta-information record. Frame applets could have more elaborate user interfaces that further interact with the user before invoking the servlet.

```
<AI:FRAME>


 <AI:FRAME-NAME>Display</AI:FRAME-NAME>

 <AI:PUBLICATION-INFO>
 <AI:HTML-UI>Display.html</AI:HTML-UI>
 </AI:PUBLICATION-INFO>

 <AI:APPLIES-TO-MIMETYPE>
   <AI:EMBEDDED>text/html;application/postscript;application/pdf;image/* </AI:EMBEDDED>
   <AI:HREF>text/html;application/postscript;application/pdf;image/* </AI:HREF>
 </AI:APPLIES-TO-MIMETYPE>

 <AI:INPUT-MENU>
   <AI:DISCOVERY />
   <AI:INPUT>location</AI:INPUT>
 </AI:INPUT-MENU>


</AI:FRAME>
```

Figure 6.5. The meta-information record for the Display *frame*

The developer of the Continuum frame applet writes the user interface. He can also opt for automatically generated user-interface code for collecting attributes values from the user. This code is generated by the Continuum server when the applet is published. For instance, the dialog boxes that query the user location for the *Display* frame in Figure 4.1 are generated due to the *AI:INPUT* XML tag in the record from Figure 6.5. The user will see these attributes as input fields, but if their values are known, the applet runtime at the browser will automatically fill them in. The user has the opportunity to initialise, correct, or update them.

Similarly, the developer can also opt for the inclusion of a user interface for choosing one service when the frame finds several to be suitable. For the *Display* frame, this is specified by the *AI:DISCOVERY* XML tag in Figure 6.5. After obtaining the results of discovery from the servlet, the user interface cycles through the found services. Only three services at a time are displayed as buttons labels. Buttons are used because some browsers on the mobile devices used had no capabilities to alter HTML lists in place.

```
<TR><TD>

<IMG SRC="http://<AI:ADAPTIVE-INFRASTRUCTURE/>/images/display.gif"
BORDER=0 WIDTH=16 HEIGHT=16 alt="display on nearby display"
OnClick="javascript:gatherInputAndCallServlet('Display','<AI:MENU-NAME/>', '<AI:URL>','<AI:MIMETYPE/>');">

</TD><TD>

<A href="javascript:gatherInputAndCallServlet('Display','<AI:MENU-NAME/>', '<AI:URL/>','<AI:MIMETYPE/>');">Display</A>

</TD></TR>
```

Figure 6.6. The HTML template used to instantiate applets for the Display *frame*

Once the complete applet code is generated, it is added to the document to which it refers, or which links to the object to which the frame applet refers. Both applets and servlets rely on a fixed runtime. To illustrate what JavaScript methods and other interface artefacts, such as toolboxes, are part of the runtime for applets. as well as what Java methods make up the runtime for servlets, we describe several of the prototyped frames. We discuss a *Print* frame (Figure 6.7), the *Display* frame from Figure 4.1, a *StoreHome* frame (Figure 6.8, Figure 6.9), a *Bookmark* frame, the *ImageAdaptation* frame from Figure 6.1, a *Next/PreviousPage* frame (Figure 6.10) for navigating transcoded Postscript documents, and the *ExchangeRoleAssertion* frame from Figure 5.4. When appropriate, we note peculiarities associated with each frame. The general functionalities of these frames are described in Chapter 5, for the *ExchangeRoleAssertion* frame, and in Chapter 3, for the rest. (To avoid clutter, all figures, except for Figure 5.4 and Figure 6.1, show only the successive screens of interacting with the respective frame, without including the initial toolbox containing it; that toolbox is similar to the toolboxes in Figure 6.3.)

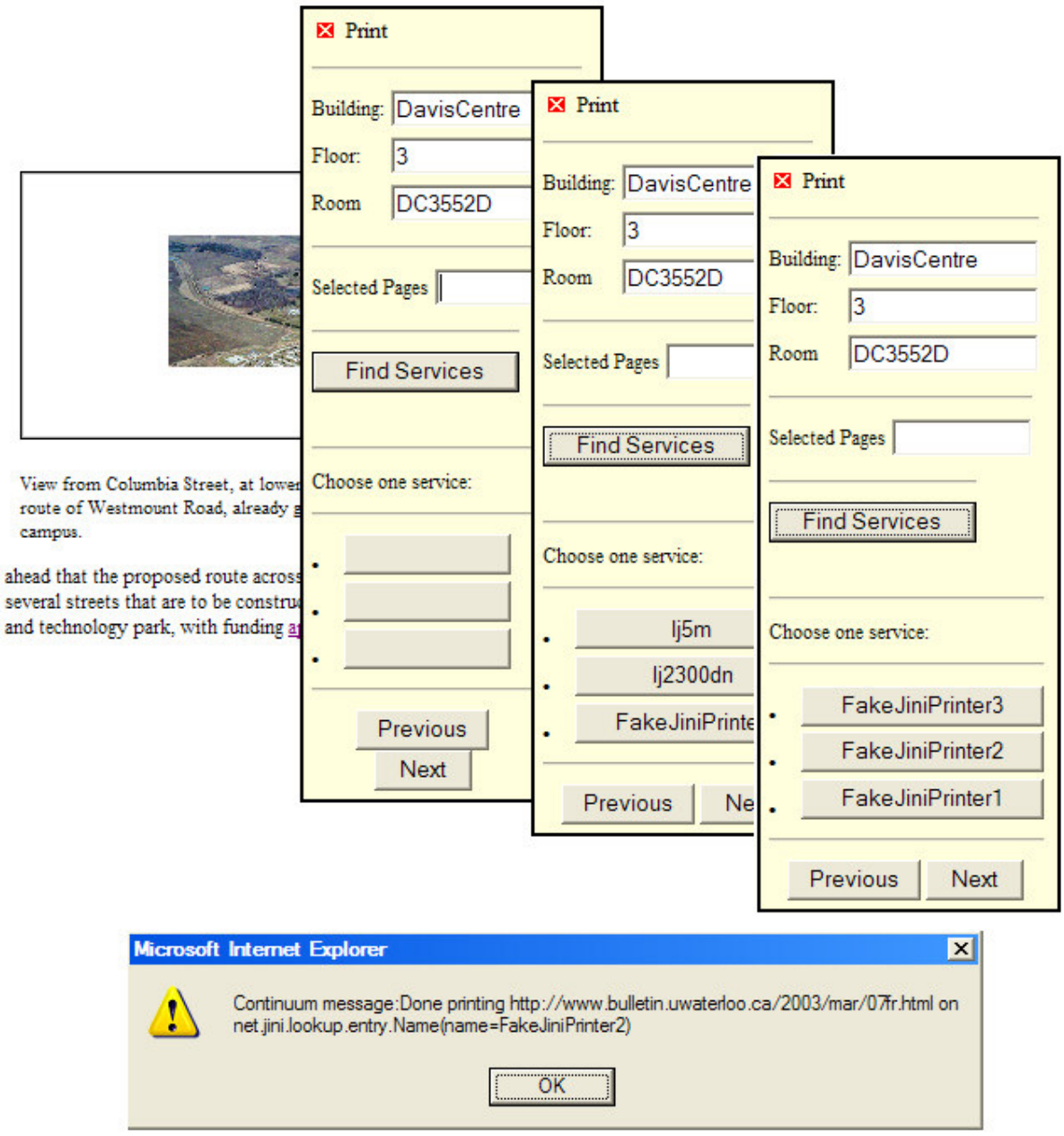

Figure 6.7. The user prints the original image, after selecting a nearby printer

The *Print* frame prints on a nearby printer. The printers are well-standardized services, yet their properties are described differently from one service-discovery protocol to another. The prototyped frame is aware only of Postscript printers, advertised over Jini [120] or SLP [100]. The frame accesses SLP printers over the line-printer daemon protocol [94]. Jini objects representing printers are accessed through a custom Java interface. We do not use the *javax.print.PrintService* interface, but it would be trivial to do

so. Since the frames handle only Postscript printers, HTML documents and images are converted to this format using specialized methods offered by the Continuum runtime, which in turn rely on off-the-shelf UNIX distilling command *html2ps*.

Unlike printers, in spite of a very well standardised hardware interface and clear semantics for the user, displays are less standardized as network devices. The closest approximation is exposing the display to which the computer is attached as an X server. The *Display* tool in Figure 4.1 can locate only displays advertised over Jini. The prototype uses a custom Java object to drive remote displays over the X Window protocol. There is no standard display template defined in SLP, hence the frame does not look for such services.

The *Print* and *Display* frames do not need to access the user's home node. Two others, *SaveHome* and *Bookmark*, do. *Bookmark* is just a specialisation of the *SaveHome* frame so we discuss only the latter. Figure 6.8 shows how the *SaveHome* frame is used. In Figure 6.9, the user provides the wrong credentials or the wrong home node to the same frame, and an error is issued. (The location of the home node is an attribute of the device.) In Figure 6.8, after the user verifies the address of the home node and enters the path where the image is to be saved, he is challenged for credentials. The figure contains an instance of the issue mentioned in Chapter 5 of current browsers not being able to convey to the user which server is being accessed. Due to Continuum running as an HTTP proxy that hijacks the requests from the browser, the password box shows www.bulletin.uwaterloo.ca, but in fact, that web server will not see the credentials. Instead, Continuum will pass the request on to the user's home node. The partial capture of a terminal window shows the stored image on the home node (a computer running Linux).

The *ImageAdaptation* frame (Figure 6.1) has been discussed sufficiently in the previous sections, so we do not restate its features here. Its applet dynamically alters the document to which it is added. Its servlet calls transcoding methods provided by the Continuum runtime. These methods are built on top of the image conversion utilities in the *ImageMagick* package, off-the-shelf UNIX software.

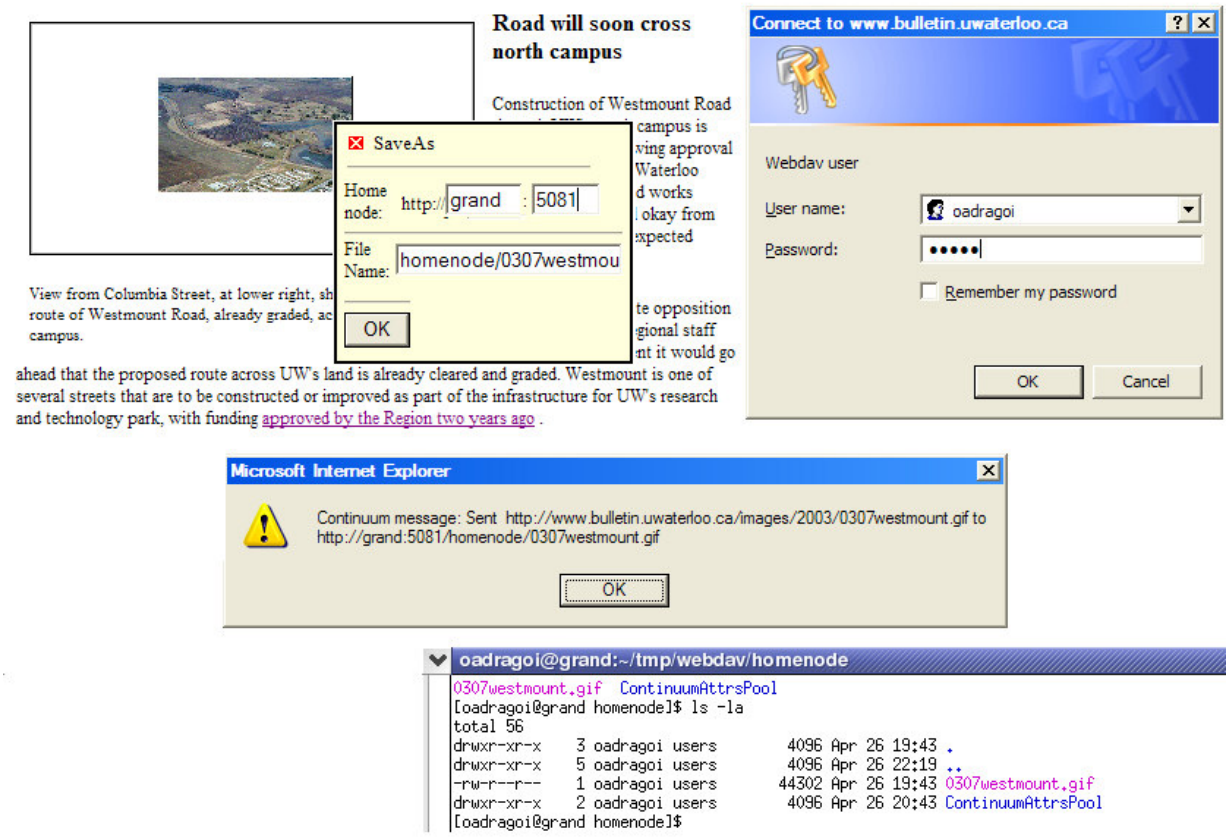

Figure 6.8. The user saves an image to her home node

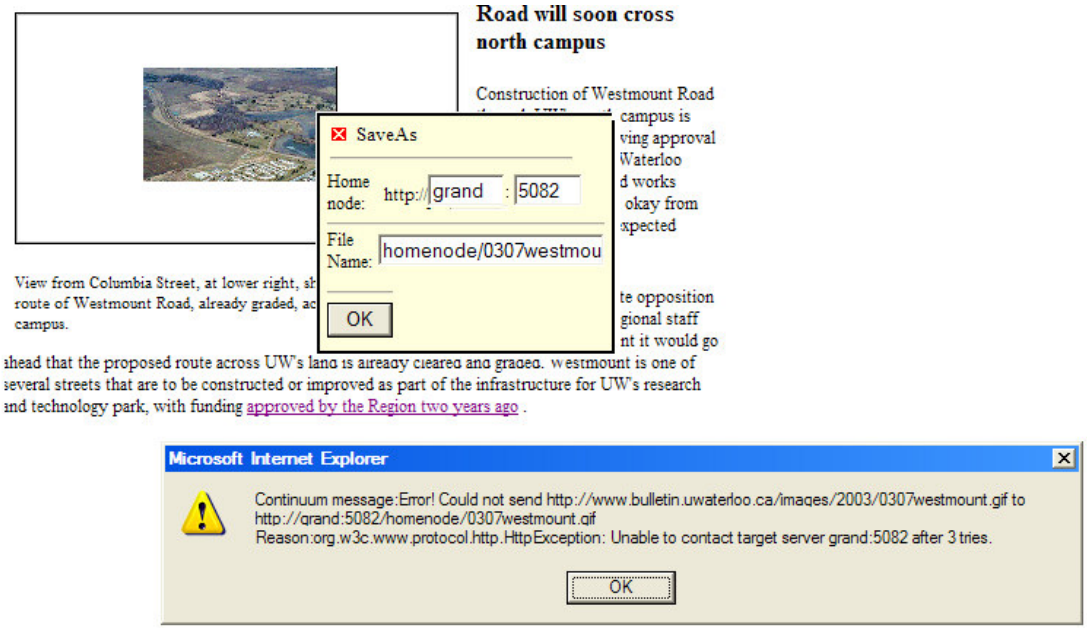

Figure 6.9. Dealing with errors in the StoreHome *frame*

The *Next/PreviousPage* frame for navigating a transcoded Postscript document is an example of a frame published only for particular hotspots. Figure 6.10 shows the functionality of this frame. Unlike the other frames, the applet of this frame is statefull. JavaScript code in the applet remembers the page number that is currently displayed as a GIF. When the forward or backward control is chosen, the page number is incremented or decremented, and then a query is sent to the servlet. The servlet extracts the correct Postscript page and it converts it to GIF with the help of conversion methods offered by the runtime. These methods use the off-the-shelf UNIX utilities *psselect* and *pstogif*.

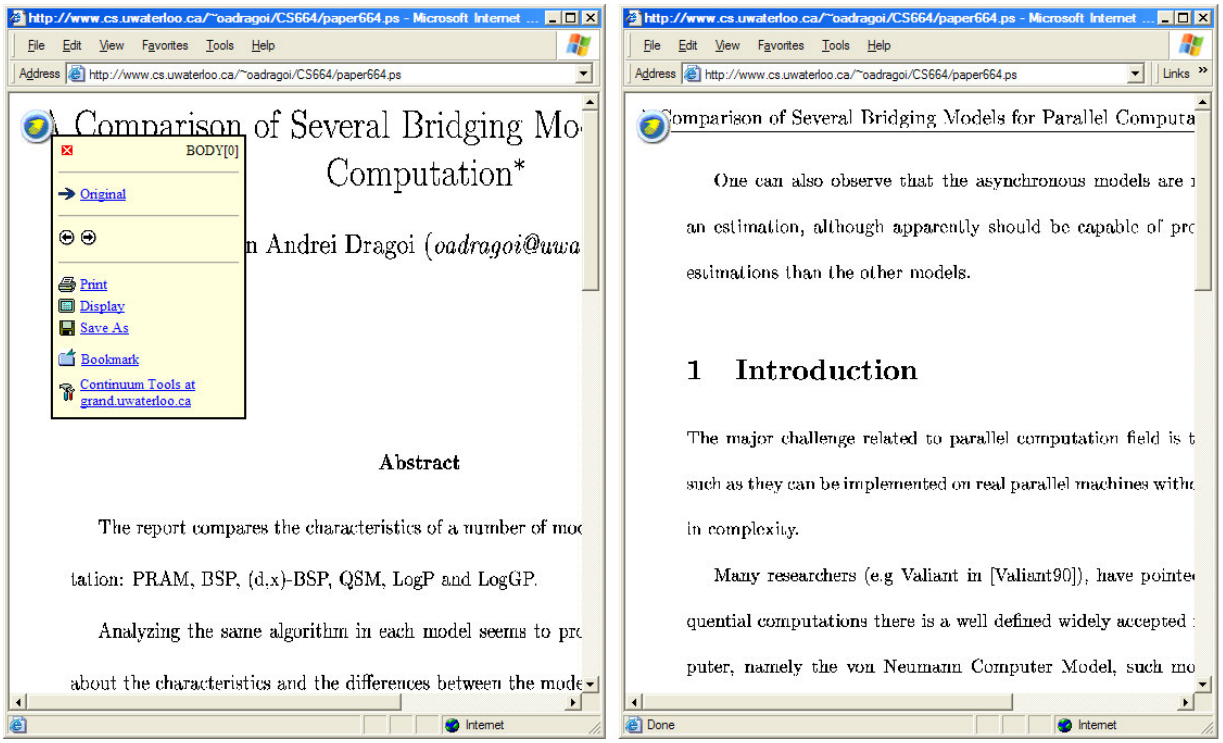

Figure 6.10. Frame to navigate a Postscript document that was transcoded to a one-page GIF

The last frame we mention here is *ExchangeRoleAssertion*. This frame calls Continuum runtime methods that store the attributes at the user's home node. For instance, the role issued in the exchanges of Figure 5.4 is stored at the recipient's home node, as the file *AIRC.Guest@HotelName.com*, in the special directory *ContinuumAttrsPool* where all attributes are kept.

There are several simplifications in the *ExchangeRoleAssertions* frame prototype. The generation of the assertion happens within the servlet (although the issuer is queried on its home node credentials), no encryption is used, and the assertions are not signed. In Chapter 5, we mentioned the more fundamental issue of generating pseudonyms for assertion holders, and the management of the public/private key pairs for both identities and pseudonyms (pseudo-identities). These issues are outside the scope of the proposed general-purpose framework for exchanging roles.

Having presented how several prototyped frames are implemented, we now summarize the runtime support Continuum offers to frame developers. For frame applets, this includes simple HTML and JavaScript templates for UI artefacts, including forms for collecting default attribute values and displaying lists of discovered services. It also includes methods for manipulating the attributes stored/cached within the browser and within the Continuum session. At the browser, all attributes (including role assertions) are kept in a cookie for the root URL of the Continuum server web domain. JavaScript that needs to access these attributes runs under the domain of the web server of the particular page viewed by the user, and JavaScript security restrictions prohibit accessing cookies for a different web domain. We use a work-around that involves a hidden HTTP request to the Continuum server, which in turn returns a page that saves the desired attribute(s) in the Continuum-domain cookie. Unfortunately, this extra roundtrip increases the latency, so the methods should be used with care. This workaround seems to be a standard procedure in web programming, so we do not detail it here. Various browsers limit the length of a cookie (typically 4K), which in turn limits the size of the attribute pool at the browser. This limitation could be worked around with multiple cookies for different paths within the Continuum domain, but since this does not change the basic functionality, we have not prototyped it.

Applets and the servlets can also use their respective runtime support for standard applet-servlet communication. This has the advantage of facilitating automatic servlet-request dispatch and automatic maintenance of the attribute cache in the session. Servlets and applets can also rely on methods to access the attributes in this cache.

For servlets, the runtime offers support for service lookup. Helper methods accept specific discovery queries and run them according to the particular discovery protocol, returning the results. Servlets have access to the known attributes of the user/device and can build queries to be run by the Service Discovery Manager on the service directories managed by Continuum, or on the service directories whose URLs have been found in role assertions, as discussed in Section 5.1.

The servlet runtime also offers methods for transcoding images, HTML to Postscript, and Postscript to HTML plus GIFs. More important, simple frames can rely on the automatic publication decision taken by the frame-publication engine, based on the information in the frame meta-information record.

6.4. Prototype limitations and future work

We end this chapter about the proof-of-concept Continuum prototype with a short discussion of the limitations and an account of future implementation work.

Currently, the frame registration facilities are minimal to non-existent. This is partly because more needs to be understood about when and how frames need to be registered. Similarly, frame publishing uses a simplistic algorithm (cycling through all the known frames). More publication heuristics need to be prototyped. In particular, experiments need to be done with heuristics that interface with additional mechanisms to capture the *a priori* context. As we have previously discussed, a short-term cache of previously discovered services together with their attributes would also improve lookup times from within the *onPublication()* method. This happens at the expense of accuracy, as some cached services might no longer be valid and some relevant service might not have been discovered previously. Currently, we have prototyped only frames that decide on their relevance without making discovery queries. Those that do need external services, such as *Print* and *Display*, launch discovery queries only when and if the user chooses to interact with them.

Prototyping more frames with innovative functionalities and experimenting with more complex attribute validation and access policies could provide new insights and new uses for role assertions.

This concludes the presentation of the proof-of-concept prototype that partly realises the Continuum design described in the previous chapters. The following chapter gives a critical evaluation of the entire architecture and design.

Chapter 7

Critical evaluation of the Continuum architecture

In spite of the recent soaring interest in ubiquitous-computing research, the field is in many respects in its infancy. Systems research in the area builds proof-of-concept prototypes to understand how to structure systems and software implementing the paradigm. User-interface research looks for innovative ways for the user to interact with such systems. Artificial-intelligence research tries to understand the kind of intelligence needed in the system to improve the user's experience. Hardware research seeks to build devices that blend smoothly into the user's everyday life. Finally, cross-disciplinary efforts try to apply the paradigm to solve particular problems in specific application domains, and to understand the social implications of ubiquitous computing.

Continuum proposes an alternative way to structure ubiquitous-computing systems and software, requiring no changes to clients and allowing for easier prototyping and deployment. The ubiquitous computing functionality is made orthogonal to web-based applications. A middleware proxy dynamically selects the frames that can mediate interactions with contextual services, and injects these frames in the web pages in transit to the mobile.

This chapter evaluates Continuum. After several examples, we show how the architecture alleviates some of the shortcomings in the current state of the art. This is followed by a description of the benefits of the approach to SPs, ISPs, content providers, and software developers, and by a short recap of how Continuum enable interactions beyond one administrative realm. The chapter ends after evaluating the prototype implementation and presenting possible future work.

We start with a set of simple examples that illustrate how Continuum effectively enables an ecology of ubiquitous-computing services at the edge of the network. A first example shows how role assertions provide bootstrapping points for service discovery. A user visiting an institution A parks her car in a

nearby lot owned by company B (different from A), and paysfor one hour using her mobile phone. The user accesses the internet using the cellular network of an ISP company C (different from A and B). While inside the institution A, she would like to extend the period she is allowed to park on the lot of company B. Normally, the software at the mobile has no way to know where to look for B's contextual services or that the user is interested in those services. With Continuum, when entering the parking lot, the user acquires a role assertion that identifies her as a parking user for the day. The assertion also contains the URL of the service directories where B advertises services. While the user is on the premises of A, the middleware run by C can find and access the service directories of B, hence publishing a tool for extending the parking period. This assumes the parking services are of a standard type. Hence, C does have the needed frame code. Since the user is a guest of A and she has acquired a role assertion to that effect at the gates, the middleware can also publish tools for the services offered by A.

A second example highlights the role of the Continuum middleware in managing software to facilitate ubiquitous computing interactions. Imagine a user is visiting university D, first without Continuum. As a guest, he gets access to the campus WiFi network. Assume that D advertises all their printers over Jini,, and that the software on the user's personal device cannot discover and interact with Jini printers, even if it can discover printers over SLP and can talk with them over LPR. In the best case, the user needs to update manually the component responsible for printing on his mobile. In the worst case, the user has to update manually all the programs installed on the mobile (if such updates are available). With Continuum, there are two possibilities: the middleware is run by D or by a user-preferred provider (E). In the first case, the middleware at D has frame code to access services offered by D. In the second case, E is a large company that keeps their frame-code repository in sync with the latest standards, including Jini. Even if D's Jini printers do not follow a standard, given that D is a large service provider in the service area of E, eventually E will acquire code for frames that can access those printers, potentially when a customer of E explicitly requests such functionality.

A third example illustrates how frame-level granularity and having frames interfacing directly with the user can facilitate ubiquitous computing interactions. Consider a user in an area where there are no printers, only a fax machine, and who wants to print the document she is accessing. Without Continuum, the ubiquitous-computing program on the user's mobile will have components that look for contextual printers. If the designers of the program did not consider that a fax can act as a printer, the user will not be able to print, even if these components know the discovery protocol used to advertise the fax. This can be solved only by updating the components on the mobile device to a different version that would also print on a fax. This update is possible only if the software developer did indeed release such an updated version. If the ubiquitous-computing functionality is provided by Continuum, the middleware will publish a frame that can fax a document. The user can make the conceptual connection that this equates printing. Of course, such a frame might not exist in the frame-code repository at the middleware, in which case, the frame code can be added to the repository without modifying the initial frame that prints only on printers, and without modifying the software at the mobile.

Let us now see how the Continuum architecture fares when compared with current state of the art. Ubiquitous-computing interactions are possible today, but often, systems research in the area concentrates on well-equipped smart spaces created to accommodate specific activities known at system/application-design time, for instance, delivering a talk. Examples of projects include Easy Living [10], iRoom [53], Aura [116], and Gaia [103]. For the administrative entity that installs and runs such a space, it is not enough to set up the electronic devices (in the case of a conference room, for instance, these devices are projectors, displays, speakers, and so on). It also has to install the software infrastructure to support ubiquitous computing interactions. Such infrastructure is often called a meta-operating system, e.g., Gaia OS or iROS.

In contrast, by design, the deployment of the Continuum middleware is not the responsibility of the SP or of the prospective users. The middleware provider is a distinct player. For example, the user's internet service provider (ISP) can act in this function. Deploying services that obey one of the widely

used standards for both access and discovery, small SPs, such as a street-corner store, can avoid any other setup other than advertising the service in a service directory. Continuum can then mediate interactions without the assumption of special relationships between these providers and the user. We have already explained how the use of non-standard services or service discovery protocols requires some cooperation between SPs and the middleware provider. Usually, this is limited to the former registering frames in the frame-code repository of the latter.

The smart space approach alone results in lack of interoperability across different space types, although smart spaces built on top of the same meta-OS usually do support the mobility of tasks and user applications from one space to another. For instance, Aura strives to bind a moved task to equivalent resources in the new space [116], but a task is not a notion in, say, iROS [53]. That is, this is not a matter of incompatibility at the level of the low-level technologies used, but at the level of abstractions employed at the meta OS level. It is unrealistic to assume all administrative entities that run smart spaces will opt for the same platform and that all users will opt to define their tasks following the same conventions and formats. Therefore, dedicated software for a specific application domain (medical, surveillance, warehouse inventory, etc.) written for one meta-OS cannot be ported directly to another, even if both use the same software platform, say Java.

Moreover, when the user moves to a smart space built on a different platform, she will have to change or add to the software used on the personal device, or at least change the way of interacting with the new space. In addition, the user in one smart space cannot access a service or a component from another space. Due to the nature of the smart-space approach, applications will not look for services outside the one smart space for which they where created, and especially not in smart spaces built on top of a different technology.

Since the interaction metaphor is part of the middleware, not at the discretion of the SP, Continuum assures consistency as the user moves from one setting to another and as he switches from one personal device to another that supports a web browser. This consistency comes from using a set of middleware

providers the user favours. This thesis does not investigate the case of users with special needs, such as blind people. They will likely use special-purpose devices and special-purpose middleware, but one could imagine this middleware being built with an architecture similar to Continuum.

Smart spaces are usually tied to specific usage scenarios (e.g., a room could be designed to be used for presentations and meetings), and while arbitrarily complex computations and interactions are possible, the nature of these computations has to be considered when the space is built. This means high maintenance costs to extend the smart space for new applications and activities.

By considering the services as a fundamental abstraction, and by mediating access interaction with them through dynamically published frames, Continuum eliminates these issues. This facilitates moving ubiquitous computing beyond controlled environments and incremental deployment, independent of the service infrastructure.

With Continuum, when a user is in an office environment, the middleware will publish more tools than when the same user is, say, in a small cafeteria, because more services can be discovered. Similarly, the middleware might publish more tools for a user holding more role assertions, because it will look for services in more service directories. If the user is in an area with many service types that are unknown to the middleware, fewer frames will be published. All these variations happen without the user having to change or manage the software on his mobile.

Currently, if a user a user encounters a service that is unknown to the software on his device, the only way to interact with that service is to discover it manually, if at all. Then, if and only if the service has a web interface, the user could browse its associated web page. This could mean the user is enters the service URL manually in a web browser. The user would also probably have to save the data object she wants to send to the service, convert it to a format the service knows about, and upload the file to the service, follow by a download of the result and by saving the result as another file. With Continuum, if the middleware has a frame for using that service, the operation can be performed just by choosing to interact with a frame attached to the document.

Continuum can coexist and interoperate with existing smart spaces, without changes to them. Specific frames can be created and published to act as gateways to existing local smart spaces or to specific functionality within those spaces, exposed and accessed as a service. Continuum can not only be used to create dedicated solutions, but also to leverage scattered devices and services already in place.

The latter can be compared to current overlay solutions that attempt to provide access to ubiquitous services of particular service types. For instance, if a hotel wants to provide its guests with access to the printers it owns and to other printing services of SPs in the hotels' geographic area, the PrintMe product [93] requires the hotel to install specific hardware provided by the overlay-middleware company. PrintMe supports only printers as service type. For other types, presumably the hotel has to wait for the overlay-middleware provider to upgrade its firmware, or it has use another overlay-middleware provider. Moreover, to print, users must agree to use the same overlay-middleware company as the hotel and they have to choose the PrintMe URL explicitly that corresponds to the particular printer on which they want to print. Presumably, if a new service type were to be being supported, the user would have to be instructed what new URLs to visit to access services of the new type.

In Continuum, users do not need to deal directly with service URLs. The role assertions the user acquires provide the middleware with bootstrapping points for service discovery, so it can automatically compile an approximation of the electronic context. Continuum enables access to any service types that the frames in the frame-code base can use.

This enables interactions with ubiquitous services of multiple providers in ways not necessarily foreseen at the design time of the middleware. The user is in control, to use the frames as it suits her. She can also sequence the operations of a set of frames. Subsequently, the middleware provider might add an additional frame that combines the manually sequenced interactions in one operation.

It is important to note there are several ways the Continuum middleware can be used.

- One is by a general provider that maintains a frame code repository so that its user base can access the widest possible set of services available in its area of service, regardless of who offers those services. An example is a large cellular ISP.
- Another is a middleware provider that targets a specific set of activities happening mostly in its own realm, but, to the best of its capabilities, also seeks to support user interactions with services outside its realm. An example is a hospital, with its specific functionality needed by the medical staff.
- A third is the specialized provider that does not support access to any services outside its realm, but chooses the Continuum architecture as a way to implement its custom ubiquitous-computing functionality, because of the benefits it offers in terms of expansibility and manageability. Examples could be a military facility or a special-care retirement home.

The Continuum architecture can be deployed in a consistent manner in a variety of settings: an airport, a home, an office, a mall. This flexibility comes partly from the fact that the application is not a technical notion at the middleware level. Only the services and the frames that drive them are.

Some of these settings might implement complex functionality that keeps the site running. Support for complex applications that require unassisted coordination of a large number of computing devices, all purposely deployed by the same service provider, is outside the scope of the Continuum architecture. They are better served by other approaches. It can be assumed that such functionality is partly exposed to users as individually accessible services. For instance, an airport might provide a service to route passengers in the fastest possible way from the arrival terminal to the terminal of a connecting flight. Users could access this service through a frame published dynamically. The frame can be published, say, due to the "traveler in transit" role a user acquired at boarding time.

Some services are not necessarily interesting for ubiquitous computing or to Continuum, because they do not need discovery. Typical examples are services available to remote users regardless of their position or the state of the physical environment, e.g., the user's bank, the search engine preferred by the user, the

security control panel for the user's house. If needed, they can be treated the same way as other services, simply by considering the user's Favourites/Bookmarks store as another service directory to check.

We continue now with a discussion of how the Continuum middleware benefits SPs, ISPs, content providers, and in particular software developers. The middleware is helpful to SPs because it can link them to potential customers that might access their services. With the tools published by Continuum, when they need to print, the guests in a hotel can automatically find out about and use the printing services of a copy store nearby. Similarly, visitors in a town can access the information services and other electronic services available to them. In a store, roles assigned to returning customers could grant them preferential prices compared to walk-in customers. Of course, this relies on the notion that middleware mediates access to all accessible services, regardless of their provider. This is not such a strong requirement as it might seem. While there is the possibility that the middleware provider would abuse its role, in the end, the users are the ones that can choose one or another middleware provider based on their level of satisfaction.

Continuum also benefits ISPs through increased revenue. This can come from service-access brokerage fees, and, more important, from an increased customer base. For instance, customers can be charged a flat fee for internet service enhanced with easy access to ubiquitous services.

The recent excitement around free internet access points confirms that ISPs have to offer additional functionality to maintain their customer base. Even if the free access points also offered Continuum-like functionality, there are obvious security and privacy concerns. Moreover, to recover their expenses such free providers might also resort to phishing through frames. A paid ISP is less likely to do that, and user's privacy might be guaranteed by contract.

In terms of processing and network load, the impact of running Continuum on the ISP acting as a middleware provider cannot be accurately estimated at this time. Due to the novelty of the paradigm, factual data is not available on the real extent of ubiquitous computing interactions. It is expected that the total number of ubiquitous services accessible to the user will be larger than what is currently available in

distributed systems or on the World Wide Web. However, it is less clear with how many such services a user will typically interact at any given moment. Nevertheless, given the way Continuum functions, if the load in terms of number of users becomes a problem, a standard load-balancing scheme can be employed. Even simple solutions, such as assigning smaller groups of users to individual workers in a pool of Continuum servers, could be effective. The processing load in terms of the number of candidate frames examined for publication can be reduced by better heuristics. Given that the synthesised electronic context is an approximation, the load can be further reduced by trading accuracy for workload reduction.

Continuum also simplifies the task of content providers, since it is conducive to separation of content from the actions that can be applied to it, and from how this content relates to ubiquitous services. Currently, many web pages provide functionality akin to frames through buttons like “send to a friend” or “show a printable copy” (this functionality does not necessarily use external services and it is used here as an only example). Such functionality is often duplicated across content-providers, and the user is confused with inconsistencies in UI.

The current tendency on the web to separate the content from its presentation confirms this. The move from HTML to XML documents opens new opportunities for frame-publication heuristics and for attaching hotspots to relevant portions of the document. While the prototype uses only HTML-tag analysis for adding hotspots, the decisions on where and what kind of hotpots should be added could capitalise on the XML tags in the document and on their semantics.

In addition to SPs, ISPs, and content providers, Continuum also benefits software developers, offering them a clear abstraction to work with. The design of a Continuum frame starts with identifying the particular innovative kind of interaction the frame will offer to the user. The developer has to choose which type of user interfaces it makes sense to support (e.g., HTML and JavaScript, WML, voice), and either make the frame robust enough to support several of them or adhere to middleware standards that would allow automatic interface generation. The Continuum design does not consider WAP or other request-reply protocols than HTTP. However, the similarities between the semantics of WAP’s Wireless

Session Protocol and HTTP indicate that the former should also be easy to support. The proof-of-concept prototype considered only laptops and PDAs. Nevertheless, current cell phones, iPods, and internet-connected approach the same kind of software environment as the former.

Designing a frame also means identifying the high-level semantics of the ubiquitous services needed, meaning the high-level service types, independent of a service-discovery protocol. For instance, a frame for delivering a presentation (an example widely used in ubiquitous-computing research) needs a display that can show the presentation and a device that can render audio. In a more basic form, the same functionality could be achieved with several frames. The first frame would let the user render the document on the projector or the display. If audio were not supported by the display, another frame would allow the user to send the audio track of the presentation to a sound system.

The developer also has to decide the hotspot types, including the data-object types, to which the frame applies, and the data-format conversions it will support. The middleware frame-runtime might implement some converters that frames may use, but in the general case, the frame will have to locate conversion services or will actually implement the conversion itself.

Next, the developer has to characterise the ubiquitous services technically. For each external service needed, one has to decide what discovery protocols are used most commonly for advertising that kind of service, and then choose which the frame will support. Then, the developer has to decide which of the standard service attributes are relevant for the frame, and what their desired values are, so that discovery queries can be formed. When more service discovery protocols are supported, the frame implicitly defines how two services advertised over different protocols are equivalent, in the sense of being usable one in the place of the other. An example is, what SLP discovery query and what Jini discovery query yield services that are comparably close to the user.

At this point, the developer also has to decide under which conditions the frames should be published. The built-in frame-publication algorithm in the target middleware might be sufficient, but often a developer will find that customised heuristics are needed. Coding such an algorithm in the frame includes

establishing what input is relevant for an informed publication decision, and how this input can be obtained from the frame-publication engine or from services that capture specific elements of the *a priori* context. Examples could be the document type, certain keywords included in the document, specific user attributes, input from context widgets [26], or from environment monitors [59]. In the prototype, only the user attributes cached in the session pool are used.

As previously noted, the publication decision is based on instantaneous contextual information, so the published frames can become out of date. This is part of the model and it is manifest to both the user and the developer. This is no different from what happens in any ubiquitous-computing software. It is just another instance of the well-known trade-off between how often software receives context updates and how stable it is.

Once all these aspects have been clarified, the developer can code the core functionality of the frame and the specific functionalities, discussed in Chapter 4. Without necessarily describing novel interactions, Table 7.1 illustrates the spectrum of interactions that the Continuum approach can support. Each entry in the table outlines the distinctive features of a frame, related to how and why the frame is published, to how the frame functions, or to its applicability. The external services required are also listed. In some cases, the publication engine and/or the frame might need to know about them at publication time. Several slightly different semantics are possible for each frame, depending on the way the frame is implemented. This is typical of Continuum, and it allows middleware providers and software developers to differentiate themselves from one another by tuning the frames to their user base.

How general or how specific a frame is remains at the discretion of the frame developer. Some of the features of the frames could overlap. Common features can be grouped in libraries that can be linked dynamically or statically. We envision that as more is understood about the dynamics of ubiquitous computing, more functionality will be built into libraries and into the middleware runtime, for frames to rely on it.

Table 7.1. Potential examples of frames

frame	distinctive feature	external services needed
PayForParking	It is published when there is some indication the user is using the parking lot: she acquired a role assertion at the gate, or she is physically on the grounds of the parking lot; two-way authentication will be required, so the user can verify he is paying for the right lot.	local parking meter and user's bank, unless some alternative scheme is used, such as credit card, e-cash, or coupons
AcceptDiscountCoupons	The user holds a *frequent-buyer* role assertion and he is in the store.	back-end corporate coupon-dispenser
ScanSurroundings	It scans the surroundings of the user to find all the nearby devices; it could be used as an office-inventory tool; the frame is not associated with a document. The frame will probably need access rights to user's devices that are beyond what JavaScript or an applet in a web page get. This frame cannot be implemented in current browsers.	other devices; they must first be advertised somehow
ControlAirConditioning	This frame does not need to be associated with particular data. Depending on the authorisation scheme implemented by the air conditioning controller, the user might have to be in an area where she can control the temperature or have an authorised identity. Essentially this frame will work as a façade for the air conditioning controller.	control-air-conditioning service
WhyAmIViewingThisPage	such a frame could be published in conjunction with each page for users with special needs, like Alzheimer patients. It could provide a brief reminder of the browsing history, in a form meaningful to the particular user; the publication of such a frame could be determined by a special role assertion that the user device holds.	assistance service for the Alzheimer patient, which keeps track of the history of the user's interactions; it could run at the user's home node
PrintThisHTMLTable	The hotspot is associated with a sub-region of a document.	printer, Postscript converter
CallThisNumber	A portion of the text in the current document is a phone number and a hotspot has been associated with it; this implies that the frame can recognize telephone numbers.	phone nearby
TranslateThisWord	The page is in a different language than the user's preferred language; the hotspot could be a floating hotspot simultaneously associated with all words in the document.	certified translation service offered by a tourism bureau

frame	**distinctive feature**	**external services needed**
InsertPhotoHere	When a photo-source device is near the user, such a frame could be published in conjunction with an editable document; unlike the other frames mentioned, this frame would modify the document; the insertion point would be made clear by the frame user-interface; this frame would need to understand the semantics of the web application it extends with ubiquitous computing functionality, so it is not a general-purpose frame.	a wireless digital camera that provides the photo or other nearby picture-source, the web site or the WebDAV server where the main copy of the document to be modified resides
HowDoIGetToThisExhibit	Such a frame could be published when the user is in a museum (e.g., she has acquired a *museum-guest* role assertion at the entrance), and on her personal device she views a web page that refers to an item on display in the museum; the frame would help the user get to the particular exhibit; the publication method of the frame will have to figure out to which exhibit the free-form web page refers; if it cannot, it should veto its publication.	museum mapping server, museum exhibits database
InitiateMRIForThisPatient	In a hospital scenario, such a frame could allow a doctor that is browsing the medical record of a patient on a PDA to initialise his interaction with an MRI device; the frame could be complex enough to automatically record the scans in the correct patient record, without specific intervention from the doctor; the hotspot with which the frame is published would be associated with the XML tag that delimits a medical record; the frame publication-decision method or the publication engine would partially understand the medical-record XML schema; the PDA user might have a role assertion that says he is the doctor of a particular patient; since in this case privacy is a bigger issue, the middleware might be in some way associated with the hospital.	MRI device, back-end patient-data database

frame	**distinctive feature**	**external services needed**
OrderMoreOfThis	In a warehouse (the user might hold an *inventory-manager* role assertion), such a frame could be published for a hotspot associated with the page that displays an inventory-number just scanned (the page is either free-form HTML or XML, and the self-publication method of the frame knows how to recognize an inventory-number page).	supplier database, supplier's ordering service

There are several reasons why the responsibility for providing and publishing frames is better held by a third party middleware than by the SPs. First, SPs do not have an interest in presenting the services of competitors to the user. It is assumed that to the best of their abilities, middleware providers will facilitate access to all potential SPs. Second, different users will have different preferences. It is less likely that each SP will support them consistently. In the smaller-scoped context of services offered by web sites (flight booking, car rental, shopping, etc.), the peculiarity of having accounts and preferences with multiple entities is highlighted by initiatives such as Liberty Alliance [64] and WS-Federation [135]. In the ubiquitous computing world, where there are significantly more services, including services offered by tangible devices, the issue can only become worse. Having to set up preferences in every institution, city, region, or country visited should be avoided. Third, a frame developer familiar with a particular application domain can provide a custom suite of frames for that domain. Likely, a general purpose SP itself would not have such expertise.

The Continuum architecture offers an innovative way to think about structuring ubiquitous computing systems. Figure 7.1 gives a loose visual indication of the relation between the Continuum approach and a "classical" ubiquitous-computing application. The grey area highlights the equivalent functionality that has to be built into a dedicated ubiquitous-computing application, and, as can be seen, it includes the functionality implemented by the web application. The ubiquitous-computing functionality in the application would correspond to a subset of the published frames.

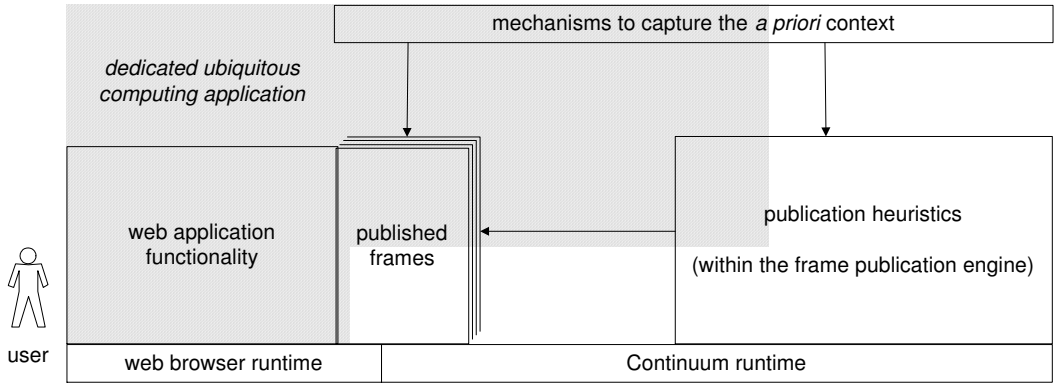

Figure 7.1. Continuum versus a dedicated ubiquitous-computing application

We have discussed how SPs, ISPs, developers, and content providers benefit from Continuum as a middleware and in particular from the notion of frames. The features in the Continuum architecture that facilitate ubiquitous computing beyond the borders of one administrative realm, across overlapping domains, are dynamic acquisition of assertions, the service-driven authorisation, frames as independent software units that can discover the services they need, and the dynamic selection of frames for publication.

The decoupling between service deployment, the development of the frames to access them, and the publication of those frames facilitates access to services from different administrative domains. Frames are generally independent of one another, and each can be written by a different developer. Since the repository of frame code is dynamically extensible with new frames, the middleware provider can eventually adjust the mix of operations in the repository to the area the middleware serves. In practice, middleware providers will probably choose to populate the frame-code base with code from a few preferred developers. The dynamics of frame-code dissemination is a complex issue that constitutes a potentially fruitful research topic on its own.

To improve interoperability further, role assertions constitute credentials in a different realm than that of the user or of the middleware provider, and they contain bootstrapping points for service discovery. The possibility of abusing such dynamically acquired credentials can be limited through the traceability of trust delegation. For instance, if a guest in the university abuses his guest rights, this can be traced to the

inviting host. She can deal with the issue through means outside the model, such as not trusting the person in the future or through law enforcement, just as one would deal with the possibility of abuse when he lends a door key to a third person.

The design of the Continuum architecture has been validated in a proof-of-concept prototype. This prototype runs under Linux and consist of ~7100 lines of compact Java 2 code. Here, by *compact* code, in reference to Java and shell script code, we mean a conservative estimate that excludes comments, blank lines, and third-party libraries and packages. By *compacted* code in reference to HTML, JavaScript, and CSS (Cascading Style Sheet) code, we mean code specifically stripped of comments, any superfluous white space, and all line breaks (no other compression implied).

In the prototype, the parsing of HTTP messages relies on message classes from the code base of the Jigsaw web server from the World Wide Web Consortium. Parsing HTML documents and HTML DOM manipulation relies on the Xerces Java Parser from the Apache Software Foundation. The SLP directory agents and interaction with them is based on an early version of the mSLP Java package from Columbia University (the mesh features are not used). The Java code uses several shell scripts (~200 lines of compact script code) that drive off-the-shelf converters included in the standard distribution of RedHat Linux, such as the *ImageMagick* package, *psselect*, *pstogif*, or *html2ps*. The code templates, from which frame applets are instantiated, total ~39K of compacted HTML, and ~600 lines (~18K) of compacted JavaScript code. The code for the corresponding set of frame servlets totals ~1400 lines of compact Java code.

Let us examine the size of the code overhead that the prototype adds. The common applet-runtime code is ~15K of compacted JavaScript code. For the document in Figure 6.1, the total additional Continuum compacted code added to the document is ~300K, out of which only ~1K is compacted CSS. The original HTML document is ~16K (typical web documents are on the order of tens of kilobytes in length) and contains many hotspots: three images, the document hotspot, and one hotspot associated with each of the 24 hyperlinks. This large number of hotspots is responsible for the large size of the extra

Continuum code. The current prototype is not optimised to use a common HTML base for the toolbox associated with each of the many hotspots in the page. In addition, applets are added independently for each hotspot. This results in redundancy. The main reason for this is that we have decided to use a smaller set of HTML features that are supported by a larger set of browsers. One way to reduce this impact is on-the-fly compression, a well-understood technique. Another way is to structure the added HTML and JavaScript code to avoid redundancy. This can reduce the overhead by at least one order of magnitude, bringing it into the range of typical document sizes. For instance, the transcoded document in Figure 6.10, which has only the document hotspot, totalled ~13.5K of compacted HTML and JavaScript code. Essentially all this code is for the toolbox and the frame-applet code. For the Continuum Tools toolbox in Figure 6.3, the added code totals ~13.5K of compacted HTML and JavaScript code. It includes applets for the frames that issue and acquire role assertions.

For the applet-servlet communication traffic resulting from generating the screenshots used throughout this thesis, the average additional payload for applet-servlet communication was ~127 bytes (by payload, here, we mean the characters in the URL, after the question mark that marks the beginning of a query; for simple examples of applet-servlet communication, see Figure 6.4). The three display services found in Figure 4.1 are returned to the client through the means of a hidden HTML file of 2.8K of compacted HTML and JavaScript code (see Chapter 6).

These numbers are included here only as a very coarse indication of the overhead added by Continuum. The richness of the user interface and the nature of the interactions in each frame can greatly affect them. At a lower level, so does making the JavaScript and HTML code more compact (e.g., shorter variable names, recoding certain blocks) or a careful choice of shorter attribute names to shorten the communication messages.

In addition to being obtained from an admittedly un-optimised prototype, these numbers are by no means statistically relevant. A statistical study requires a sizable pool of testers to interact with Continuum in a variety of situations, through frames designed to accommodate several application domains and

varied services and devices. The expertise needed in human-computer interaction and in psychology, obligatory for such an endeavour, situates a study like this outside the scope of this thesis. Nevertheless, it constitutes an existing area for future work.

Indeed, the prototype or a production reimplementation can serve as an experimental test-bed for user studies and for the implementation of frame suites for diverse application domains. This could bring a better understanding of the nature of ubiquitous-computing interactions. The generality of the Continuum approach makes it suitable for such a purpose. Here we mention again only two of the features that contribute to this generality: the electronic context is synthesised by a separate, extensible, system component (the frame-publication engine) and the contract of the frame interface is directly with the user.

We end this chapter with discussion of future work and a review of issues that were considered outside the scope of this thesis. The Continuum server functions as an HTTP proxy. It is left to future work to study middleware-server handoff and the issue of contention or composition between frames published by multiple middleware providers, say, between the user's preferred middleware provider and the ISP-run middleware. Such composition could result in distinct toolboxes from each middleware provider or in a toolbox with mixed tools that are clearly marked with their origin. Inadvertent interactions of frame applets with browser plugins or proxies installed in the browser, as well as contention with other transparent proxies or double proxies, has also been left to future work.

One area of future work is devising and prototyping more frames, in particular frames for specific application domains and frames with more complex interfaces. We also plan to experiment with different hotspots, e.g., XML tags semantics or lexical parts. More types of hotspots require a refinement of the meta-information that characterises hotspots.

Another future work area is related to prototyping more frames to help refine the interface between frame applets, frame servlets, and their respective runtimes. This might also clarify if there is a need for a standardised API for frames to communicate with each other. A distinct future work sub-area is represented by experimentation with frames with special non-graphical user interfaces.

Frames that accept data from services and insert it in the document viewed by the user, at the position marked with the hotspot to which the frame is associated, are also problematic to prototype. For web applications, such frames do not make sense unless the middleware partly understands the semantics of the web application, which is against the assumptions in Continuum. This brings us to another area of future work, which is to validate the Continuum approach with other viewers than a simple web browser and other document formats than HTML. Dealing with viewers such as Microsoft PowerPoint or Microsoft Word that access remote data over a derivative of HTTP, like WebDAV, is easier than dealing with viewers that access remote data over other protocols, such as mail readers, calendars, etc. Related, but harder, is to experiment with publishing tools in conjunction with data accesses local at the device. This would require installing certain plugins in the viewers at the mobile, to request the toolboxes. We have already discussed that this can function only with viewers that allow installable hooks to be called when each structural element of the document is accessed.

Sophisticated heuristics for frame publication were outside the scope of this thesis. There are several limitations with the current implementation of frame publication. Only the data MIME type and the type of hotspot are used in the publication decision. Taking into account other information is delegated to the frame.

Services for context collection and aggregation are also outside the scope of the thesis. More prototyping is needed of other ways to capture and monitor the *a priori* context and to use it for publication decisions, from within frames and from the publication engine. In part, this means more ways to discover service directories. Support for more realistic ways to obtain user location is also needed. In the current prototype, the middleware uses only mobile/user attributes and the discovery bootstrapping points found in role assertions. More needs to be done to improve publication latency. Caching techniques for already-discovered services could be a place to start.

Related to frame publication, this thesis did not deal with sophisticated mechanisms for frame code dissemination or with ISP roaming agreements that include frame-code exchanges. As more is understood

about how users will use ubiquitous computing and how SPs will really choose to deploy their services, specific mechanisms to extend the frame code base at the Continuum server will need to be devised.

At the level of the framework for authorisation and access, further prototyping work includes prototyping services or service proxies that authorise based on role assertions and on other combinations of attributes, as well as experimentation with different ways of validating attributes. This might require conversions among different location formats. In some cases, the mobile might have to authenticate and authorise interactions with services (two-way authentication), not only the other way round.

Mechanisms for the exchange of role assertions without the involvement of Continuum servers are also part of the future work. They would require different frame publication techniques. The UI issues for handling credentials, outlined in Chapter 6, also need a solution.

More fundamental is the opportunity for future research into what additional information the role assertions can include, to help the publication engine or the frame to make a more informed decision on its relevance. Potentially interesting examples of such information are a description of the services that accept the granted role, different ways to specify the role validity (such as the physical area of validity), or even specifying service URLs inside the assertion. There are also the connected issues of how users learn that they can grant certain roles and how users learn the semantics of those roles and the implications of granting them. Research is also needed to understand and manage the user identities to which roles are granted.

This chapter has evaluated the benefits and the shortcomings of the Continuum architecture. The next chapter concludes the thesis by summarising the contributions of this work and the way it advances the knowledge of the field.

Chapter 8

Conclusions

This thesis has presented an architecture, a design based on it, and a proof-of-concept partial implementation that bring ubiquitous computing to chaotic environments, enabling an ecology at the edge of the network, between users, service providers, internet service providers, content providers, and software developers.

The Continuum architecture makes several important contributions. It uses a novel way to structure software for ubiquitous computing, more appropriate to the specific characteristics of the paradigm. The ubiquitous computing functionality is considered orthogonal to other application logic. The former is dynamically "attached" to the data accessed by the latter, independent of the reasons why that data is accessed.

Continuum opts for software granularity at the level of self-contained frames—each a simple tool with precise functionality that helps the user interact with contextual services in conjunction with the data to which the tool is attached. Suites of frames can be defined for specific activities, and tools from several suites might be published, so that the user can arbitrarily multitask between the corresponding activities.

This yields software that adapts to the environment variability and is manageable by the system, not by the mobile user. With Continuum, the ubiquitous-computing functionality available to a particular user varies dynamically, since only the frames that can realistically function are selected and instantiated. This variation follows from the set of services that can be realistically discovered on behalf of a user given the information available to the middleware, from the amount of time that can be reasonably allocated to this discovery so that the user has a good experience with the system, and from the code base available at the middleware.

The architecture addresses the chaos inherent to the ubiquitous-computing paradigm through the notion of role assertions acquired dynamically by the mobile (the user). This enables access across administrative domains, without cooperation or information exchange among the authorities of the domains. The tools published by the middleware can interact with parts of the environment under the control of different administrative domains, because a role assertion can capture the conceptual notion of a "client" of a certain administrative domain.

Roles are not mutually exclusive. The user chooses what hat she is wearing, initially by accepting a role assertion and then by choosing to interact with a particular tool. A user could also give up a role assertion she has. From the role assertions a particular user holds, the middleware can determine bootstrapping points for service discovery on behalf of that user. The role assertions also represent credentials the tools/the mobile can use to access services in a different administrative domain.

Another contribution of the Continuum architecture is demonstrating how, without interfering with the functionality offered by other parties, software developers can make independent contributions to the user's ubiquitous-computing experience by writing a frame or a suite of frames.

The conceptual architecture has been refined to a design where the ubiquitous-computing functionality is orthogonal (as above) to unmodified web-based applications, being attached to web pages and objects embedded in them. Recursion is inherent in the design, in the sense that interaction with a frame might result in the initiation of another web application (e.g., in a new browser window). When and why this happens and how this is conveyed to the user is an issue internal to the frame.

Essentially, the middleware augments existing web applications with ubiquitous-computing functionality, without modifying the applications, without changes to the clients (simple Internet browsers), and with no need to update/manage software at the client since it is the middleware that deals with the complexity of managing the frame code base.

The design presented in this thesis explicitly includes the possibility of discrepancies between the assumptions of ubiquitous-computing software developers and field realities. The existence of multiple

administrative domains is exposed through the active involvement of the user in role-assertion acquisition. The unavailability or the inaccessibility of services is reflected implicitly by not publishing the corresponding tools (frames). Moreover, by contract, the toolboxes are best-effort approximations to the available functionality: tools to improve the approximation, by, say, soliciting additional information, might also be published by the middleware.

In Continuum, the function of the middleware becomes to synthesise continuously an *electronic context* that helps the user understand the surrounding services and use them on the data viewed on her personal devices. A frame represents the unit of this electronic contest, and the frame design identifies the functionality of the interfaces with the middleware runtime and with the user, so that the resulting system is expansible without the need to redeploy the middleware. This frame design include UI generation, directing discovery (not necessarily running it), including equivalences between service types and service attributes from different discovery protocols, directing transcoding (not necessarily implementing it), deciding on publication of a frame instance, and, of course, implementing the frame logic to drive the external contextual services. One other contribution of this design is that frames interface directly with the user through their UI.

Last, as realised in this thesis, Continuum provides an example of additional functionality that ISPs can provide to maintain their customer base and justify the access fees.

The proof-of-concept prototype implementation of Continuum shows the design can be implemented and the architecture is feasible with web applications. The prototype implements several frames that demonstrate the general frame functionality, including driving discovery queries over multiple service discovery protocols and making equivalences across discovery protocols between service types. It also implements special-purpose frames to exchange role assertions between users using the same middleware provider, contributing to the realisation of the framework for service discovery and authorisation included in the architecture.

The work presented in this thesis opens several general research areas. (A more Continuum-specific specific future work description is provided at the end of Chapter 7.)

The way we have redefined the principal function of ubiquitous computing middleware opens an entire field of research for publication heuristics to improve the electronic context approximations and to incrementally provide better support for the ubiquitous-computing user. Good publication heuristics are central to a positive user experience with ubiquitous computing. The main goal of such heuristics is to keep a balance between not overwhelming the user with too many published frames, and failing to publish useful frames. Publication algorithms that are more efficient, e.g., some sort of greedy algorithm, and the use of other elements of the *a priori* context in the decision process, are two examples. The use of select role semantics to improve frame-publication decisions, either at frame level or at Continuum level, is another. A role can capture not only the notion that a user is a preferred customer for a certain SP, but also the notion that the user is involved in a certain activity. A middleware server aware of the semantics of such a role could favour tools from a suite associated with that activity. In devising advanced publication heuristics, one could also employ usage-history tracking or determining if the user is in the service area of a discovered service, to avoid publishing a frame that eventually will not be allowed to access needed external services. Research info techniques to let a user further tune the publication process is another example.

Possibly interdisciplinary research is essential to understand better the dynamics of the business and technological relation between SPs, code developers, and ISPs to enable the best support for the users in a given area. At the technical level, relevant aspects are frame code dissemination to the middleware providers that serve the area, roaming agreements between ISPs, to preserve the user experience, the dissemination of role semantics to business partners, the design and deployment of services that issue role assertions, as opposed to users issuing them.

At the level of the theory of system design, a challenging research area is to find alternative materialisations for the artefacts that make up the electronic context, and how these compare with the

materialisations of the electronic context as a set of frames, as proposed in this thesis. Elements of comparison can be user friendliness, the scalability of the system that synthesises the electronic context, the ease of software development for the resulting platform, etc.

The Continuum architecture and the prototype, or an optimised version of the prototype, could also serve as a platform for multidisciplinary research to understand better what the needs of ubiquitous computing are, because they provision for easy addition of new tools and tool suites. This could mean user trials and usability studies to establish the most appropriate kind of interactions for ubiquitous computing and to devise innovative user interfaces.

Bibliography

[1] G. Banavar, J. Beck, E. Gluzberg, J. Munson, J. Sussman and D. Zukowski, "Challenges: An application model for pervasive computing", in *6th International Conference on Mobile Computing and Networking (MOBICOM '00)*, pp. 266–274, Boston, MA, USA, August 2000.

[2] A. Barbir, R. Chen, M. Hofmann, H. Orman and R. Penno, "An Architecture for Open Pluggable Edge Services (OPES)", Internet draft, work in progress, expires June 2003, December 2002.

[3] V. Bharghavan and V. Gupta, "A framework for application adaptation in mobile computing environments", in *21st International Computer Software and Applications Conference*, pp. 573–579, Bethesda, MD, USA, August 1997.

[4] T. Bickmore and B. N. Schilit, "Digestor: Device-independent access to the world-wide web", in *6th International World Wide Web Conference*, pp. 1075–1082, Santa Clara, CA, USA, April 1997.

[5] M. Blaze, J. Feigenbaum and J. Lacy, "Decentralized trust management", in *IEEE Symposium on Research in Security and Privacy*, pp. 164–173, Oakland, CA, USA, May 1996.

[6] Bluetooth SIG, "Bluetooth Core Specification v1.2", http://www.bluetooth.org/, November 2003.

[7] J. Böliger and T. Gross, "A framework-based approach to the development of network-aware applications", *IEEE Transactions on Software Engineering*, vol. 24, n. 5, pp. 376–390, May 1998.

[8] D. Booth, M. Champion, C. Ferris, F. McCabe, E. Newcomer and D. Orchard, "Web Services architecture", W3C working draft, work in progress, http://www.w3c.org/, May 2003.

[9] C. Brooks, M.S. Mazer, S. Meeks and J. Miller, "Application-specific proxy servers as HTTP stream transducers", *World Wide Web Journal*, vol. 1, n. 1, 1996.

[10] B. Brumitt, B. Meyers, J. Krumm, A. Kern and S. Shafer, "EasyLiving: Technologies for intelligent environments", in *2nd International Symposium on Handheld and Ubiquitous Computing*, pp. 12–29, Bristol, UK, September 2000.

[11] L. Bussard and Y. Roudier, "Authentication in ubiquitous computing", in *Workshop on Security in Ubiquitous Computing (with UBICOMP '02)*, Göteborg, Sweden, September-October 2002.

[12] L. Bussard and R. Molva, "One-time capabilities for authorizations without trust", in *2nd IEEE International Conference on Pervasive Computing and Communications (PERCOM '04)*, pp. 351–355, Orlando, Florida, USA, March 2004.

[13] A. T. Campbell, "Mobiware: QoS aware middleware for mobile multimedia communications", in *7th IFIP International Conference on High Performance Networking (HPN)*, pp. 166–183, New York, NY, USA, April 1997.

[14] G. Chen and D. Kotz, "A survey of context-aware mobile computing research", Technical Report TR2000-381, Department of Computer Science, Dartmouth College, 2000.

[15] J. Chen, B. Zhou, J. Shi, H. Zhang and Q. Fengwu, "Function-based object model towards website adaptation", in *10th international conference on World Wide Web*, pp. 587–596, Hong Kong, China, 2001.

[16] G. Chen and D. Kotz, "Solar: A pervasive-computing infrastructure for context-aware mobile applications", Technical Report TR2002-421, Department of Computer Science, Dartmouth College, February 2002.

[17] M. Cherniack, M. J. Franklin and S. Zdonik, "Expressing user profiles for data recharging", *IEEE Personal Communication: Special Issue on Pervasive Computing*, vol. 8, n. 4, pp. 32–38, August 2001.

[18] D. M. Chess, C. G. Harrison and A. Kershebaum, "Mobile agents: Are they a good idea?", IBM Research Division, T.J. Watson Research Center, RC 19887, December 1994.

[19] K. Cheverst, N. Davies, K. Mitchell and A. Friday, "Experiences of developing and deploying a context-aware tourist guide: The GUIDE project", in *6th Annual International Conference on Mobile Computing and Networking*, pp. 20–31, Boston, MA, USA, 2000.

[20] M. J. Covington, W. Long, S. Srinivasan, A. K. Dev, M. Ahamad and G. D. Abowd, "Securing context-aware applications using environment roles", in *6th ACM Symposium on Access control models and technologies*, pp. 10–20, Chantilly, Virginia, USA, 2001.

[21] L. Cranor, M. Langheinrich, M. Marchiori, M. Presler-Marchall and J. Reagle, "The platform for privacy preferences 1.0 (P3P1.0) specification", W3C working draft, work in progress, http://www.w3c.org/, May 2000.

[22] N. Davies, A. Friday, G. S. Blair and K. Cherverst, "Distributed systems support for adaptive mobile applications", *Mobile Networks and Applications*, vol. 1, n. 4, pp. 399–408, December 1996.

[23] N. Davies, A. Friday, S. P. Wade and G. S. Blair, "L^{2}limbo: A distributed systems platform for mobile computing", *Mobile Networks and Applications*, vol. 3, n. 2, pp. 143–156, August 1998.

[24] N. Davies and H. Gellersen, "Beyond prototypes: Challenges in deploying ubiquitous systems", *IEEE Pervasive Computing*, vol. 1, n. 1, pp. 26–35, January-March 2002.

[25] A. K. Dey and G. D. Abowd, "Towards a better understanding of context and context-awareness", Technical Report IT-GVU-99-29, Graphics, Visualization and Usability Center and College of Computing, Georgia Institute of Technology, 1999.

[26] A. K. Dey, *Providing architectural support for building context-aware applications*, PhD thesis, Georgia Institute of Technology, December 2000.

[27] T. T. Drashansky, S. Weerawarana, A. Joshi, R. A. Weersinghe and E. N. Houstis, "Software architecture of ubiquitous scientific computing environments for mobile platforms", *Mobile Networks and Applications*, vol. 1, n. 4, pp. 421–432, December 1996.

[28] W. K. Edwards, M. W. Newman and J. Z. Sedivy, "The case for recombinant computing", Technical Report CSL 01-1, Xerox Parc, April 2001.

[29] W. K. Edwards, M. W. Newman, J. Sedivy, T. Smith and S. Izadi, "Challenge: Recombinant computing and the Speakeasy approach?", in *8th International Conference on Mobile Computing and Networking (MOBICOM '02)*, pp. 279–286, Atlanta, GA, USA, September 2002.

[30] C. English, P. Nixon, S. Terzis, A. McGettrick and H. Lowe, "Dynamic trust models for ubiquitous computing", in *Workshop on Security in Ubiquitous Computing (with UBICOMP '02)*, Göteborg, Sweden, September-October 2002.

[31] T. Erickson, "Technical opinion: Some problems with the notion of context-aware computing", *Communications of ACM*, vol. 45, n. 2, pp. 102–104, February 2002.

[32] M. Esler, J. Hightower, T. Anderson, and G. Borriello, "Data-centric networking for invisible computing: The Portolano project at the University of Washington", in *5th International Conference on Mobile Computing and Networking (MOBICOM '99)*, pp. 256–262, Dallas, TX, USA, August 1999.

[33] A. Fox, *A framework for separating server scalability and availability from internet application functionality*, PhD thesis, University of California at Berkley, 1998.

[34] A. Fox, I. Goldberg, S. D. Gribble, D. C. Lee, A. Polito and E. A. Brewer, "Experience with Top Gun Wingman, a proxy-based graphical web browser for 3Com PalmPilot", in *IFIP International Conference on Distributed Systems Platforms and Opens Distributed Processing (MIDDLEWARE '98)*, pp. 407–424, Lake District, U.K., September 1998.

[35] P. Gauthier, J. Cohen, M. Dunsmuir and C. Perkins, "The web proxy auto-discovery protocol", Internet draft, work in progress, expires December 1999, IETF, July 1999.

[36] E. Gerck, "Overview of certification systems: X.509, CA, PGP, and SKIP", *The Bell*, vol. 1, n. 3, pp. 3–8, July 2000.

[37] Y. Y. Goland, "Simple service discovery protocol/1.0, operating without an arbiter", Internet draft, work in progress, expires April 2000, IETF, October 1999.

[38] S. D. Gribble, M. Welsh, R. von Behren, E. A. Brewer, D. Culler, N. Borisov, S. Czerwinski, R. Gummadi, J. Hill, A. Joseph, R. H. Katz, Z. M. Mao, S. Ross and B. Zhao, "The Ninja architecture for robust internet-scale systems and services", *Computer Networks*, vol. 35, n. 5, pp. 473–497, March 2001.

[39] R. Grimm, *System support for pervasive applications*, PhD thesis, University of Washington, December 2002.

[40] R. Han and P. Bhagwat, "Dynamic adaptation in an image transcoding proxy for mobile web browsing", *IEEE Personal Communications Magazine*, pp. 8–17, December 1998.

[41] U. Hengartner and P. Steenkiste, "Protecting people location information", in *Workshop on Security in Ubiquitous Computing (with UBICOMP '02)*, Göteborg, Sweden, September-October 2002.

[42] C. K. Hess and R. H. Campbell, "A context file system for ubiquitous computing environments", Technical Report UIUCDCS-R-2002-2285, Department of Computer Science, University of Illinois at Urbana-Champaign, July 2002.

[43] T. D. Hodes and R. H. Katz, "Composable ad hoc location-based services for heterogeneous mobile clients", *ACM Wireless Networks Journal, Special issue on mobile computing: Selected papers from MOBICOM '97*, vol. 5, n. 5, pp. 411–427, October 1999.

[44] K. Holtman and A. Mutz, "Transparent content negotiation in HTTP", RFC2295, Network Working Group, IETF, March 1998.

[45] B. C. Housel, G. Samaras and D. B. Lindquist, "WebExpress: A client/intercept based system for optimising web browsing in a wireless environment", *Mobile Networks and Applications*, vol. 3, n. 4, pp. 419–431, June 1998.

[46] A. C. Huang, B. C. Ling, S. Ponnekanti and A. Fox, "Pervasive computing: What is it good for?", in *Workshop on Mobile Data Management (with MOBICOM '99)*, Seattle, WA, USA, September 1999.

[47] A. Huang, B. Ling and J. Barton, "Making computers disappear: Appliance data services", in *7th annual ACM/IEEE international conference on Mobile computing and networking*, pp. 108–121, Rome, Italy, August 2001.

[48] J. Inouye, S. Cen, C. Pu and J. Walpole, "System support for mobile multimedia applications", in *7th International Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV '97)*, pp. 143–154, St. Louis, MI, USA, May 1997.

[49] U. Jendricke, M. Kreutzer and A. Zugenmaier, "Pervasive privacy with identity management", in *Workshop on Security in Ubiquitous Computing (with UBICOMP '02)*, Göteborg, Sweden, September-October 2002.

[50] X. Jiang, J. Hong and J. Landay, "Approximate information flows: Socially based modeling of privacy in ubiquitous computing", in *4th International Conference in Ubiquitous Computing (UBICOMP '02)*, pp. 176–193, Göteborg, Sweden, September-October 2002.

[51] X. Jiang and J. A. Landay, "Modeling privacy control in context-aware systems", *IEEE Pervasive Computing*, vol. 1, n. 3, pp. 59–63, July-September 2002.

[52] C. Jiang and P. Steenkiste, "A Hybrid Location Model with a Computable Location Identifier for Ubiquitous Computing", in *4th international conference on Ubiquitous Computing (UBICOMP '02)*, pp. 246–263, Göteborg, Sweden, September-October 2002.

[53] B. Johanson, A. Fox and T. Winograd, "The interactive workspaces project: Experiences with ubiquitous computing rooms", *IEEE Pervasive Computing Magazine*, vol. 1, n. 2, pp. 67–74, April-June 2002.

[54] A. D. Joseph, J. A. Tauber and M. F. Kaashoek, "Mobile computing with Rover toolkit", *IEEE Transactions on Computers: Special Issue on Mobile Computing*, vol. 46, n. 3, pp. 337–357, March 1997.

[55] G. Judd and P. Steenkiste, "Providing contextual information to pervasive computing applications", in *1st IEEE International Conference on Pervasive Computing and Communications (PERCOM '03)*, pp. 133–142, Fort Worth, TX, USA, March 2003.

[56] L. Kagal, T. Finn and A. Joshi, "Trust-based security in pervasive computing environments", *IEEE Computer*, vol. 34, n. 12, pp. 154–157, December 2001.

[57] T. G. Kanter, *Adaptive personal mobile communication. Service architecture and protocols*, PhD thesis, Laboratory of Communication Networks, Department of Microelectronics and Information Technology and Royal Institute of Technology, Stockholm, Sweden, November 2001.

[58] R. H. Katz, "Adaptation and mobility in wireless information systems", *IEEE Personal Communication Magazine*, vol. 1, pp. 6–17, January-April 1994.

[59] D. Kidston, J. P. Black, T. Kunz, M. E. Nidd, M. Lioy, B. Elphick and M. Ostrowski, "Comma, a communication manager for mobile applications", in *International Conference of Wireless Communications (WIRELESS '98)*, pp. 103–116, TR Labs, Calgary, AB, Canada, July 1998.

[60] T. Kindberg, J. Barton, J. Morgan, G. Becker, D. Caswell, P. Debaty, G. Gopal, M. Frid, V. Krishnan, H. Morris, J. Schettino and B. Serra, "People, places, things: Web presence for the real world", Technical Report HPL-2000-16, HP Laboratories Palo Alto, February 2000.

[61] T. Kindberg and A. Fox, "System software for ubiquitous computing", *IEEE Pervasive Computing*, vol. 1, n. 1, pp. 70–81, January-March 2002.

[62] M. Lamming, M. Eldridge, M. Flynn, C. Jones and D. Pendlebury, "Satchel: Providing access to any document, any time, anywhere", *ACM Transactions on Computer-Human Interaction*, vol. 7, n. 3, pp. 322–352, September 2000.

[63] M. Langheinrich, V. Coroama, J. Bohn and M. Rohs, "As we may live – Real-world implications of ubiquitous computing", Technical report, Institute for Pervasive Computing, ETH Zurich, Switzerland, April 2002.

[64] Liberty Alliance Project, "Introduction to the Liberty Alliance identity architecture, revision 1.0", White paper, http://www.projectliberty.org/, March 2003.

[65] Liberty Alliance Project, "Liberty Alliance and WS-Federation: A comparative overview", Technical White paper, http://www.projectliberty.org/, October 2003.

[66] E. Lara, R. Kumar, D. S. Wallach and W. Zwaenepoel, "Puppeteer: Component-based adaptation for mobile computing", in *3rd USENIX Symposium on Internet Technologies and Systems (USITS)*, pp. 159–170, San Francisco, CA, USA, March 2001.

[67] E. Lara, R. Kuma, D. S. Wallach and W. Zwaenepoel, "Collaboration and Multimedia Authoring on Mobile Devices", in *1th International Conference on Mobile Systems, Applications, and Services (MOBISYS)*, pp. 287–302, San Francisco, CA, USA, May 2003.

[68] H. Lei, D. M. Sow, J. S. Davis, G. Banavar and M. R. Ebling, "The design and applications of a context service", *ACM SIGMOBILE Mobile Computing and Communications Review*, vol. 6, n. 4, pp. 45–55, October 2002.

[69] B. Li and K. Nahrstedt, "QualProbes: middleware QoS profiling services for configuring adaptive applications", in *IFIP International Conference on Distributed Systems Platforms and Open Distributed Processing (MIDDLEWARE '00))*, pp. 256–272, New York, NY, USA, April 2000.

[70] Q. Li, "An architecture for geographically-oriented service discovery on the internet", Master's thesis, Department of Computer Science, University of Waterloo, 2002.

[71] M. Lioy, and J. P. Black, "Providing network services at the base station in a wireless networking environment", in *International Conference of Wireless Communications (WIRELESS '97)*, pp. 29–39, TR Labs, Calgary, AB, Canada, July 1998.

[72] J. R. Lorch and A. J. Smith, "Software strategies for portable computer energy management", *IEEE Personal Communications Magazine*, vol. 5, n. 3, pp. 60–73, June 1998.

[73] W. Y. Lum and F. C. M. Lau, "A context-aware decision engine for content adaptation", *IEEE Pervasive Computing*, vol. 1, n. 3, pp. 41–49, July-September 2002.

[74] P. Maniatis, M. Roussopoulos, E. Swierk, K. Lai, G. Appenzeller, X. Zhao and M. Baker, "The Mobile People architecture", *ACM Mobile Computing and Communications Review*, vol. 1, n. 2, pp. 36–42, July 1999.

[75] I. Maric, "Connection establishment in the Bluetooth system", Master's thesis, Electrical and Computer Engineering Department, Rutgers, The State University of New Jersey, October 2000.

[76] M. McIlhagga, A. Light and I. Wakeman, "Towards a design methodology for adaptive applications", in *4th annual ACM/IEEE international conference on Mobile computing and networking*, pp. 133–144, Dallas, TX, USA, October 1998.

[77] D. H. McKnight and N. L. Chervany, "The Meanings of Trust", Technical Report MISRC WP 96-04, MIS Research Center, Carlson School of Management, University of Minnesota, April 1996.

[78] Contributing Members of the UPnP Forum, "Universal Plug and Play, device architecture specification", http://www.upnp.org/, June 2000.

[79] Microsoft Corporation, "Understanding UPnP: A white paper", White paper, http://www.upnp.org/, June 2000.

[80] J. Al-Muhtadi, R. H. Campbell, A. Kapadia, D. Mickunas and S. Yi, "Routing through the Mist: Privacy Preserving Communication in Ubiquitous Computing Environments", in *International Conference of Distributed Computing Systems (ICDCS '02)*, pp. 65–74, Vienna, Austria, July 2002.

[81] G.J. Nelson, *Context-aware and location systems*, PhD thesis, Clare College, University of Cambridge, January 1998.

[82] B. C. Neuman and T. Y. Ts'o, "Kerberos: An authentication service for computer networks", *IEEE Communications*, vol. 32, n. 9, pp. 33–38, September 1994.

[83] M. W. Newman, J. Z. Sedivy, C. M. Neuwirth, W. K. Edwards, J. I. Hong, S. Izadi, K. Marcelo, T. F. Smith and J. Sedivy, "Designing for serendipity: Supporting end-user configuration of ubiquitous computing environments", in *Conference on Designing interactive systems: Processes, practices, methods, and techniques*, pp. 147–156, London, England, 2002.

[84] E. Di Nitto, G. Sassaroli and M. Zuccalà, "Adaptation of web contents and services to terminals capabilities: The @Terminals approach", in *1st IEEE International Conference on Pervasive Computing and Communications (PERCOM '03)*, pp. 433–442, Fort Worth, TX, USA, March 2003.

[85] B. D. Noble and M. Satyanarayanan, "A research status report on adaptation for mobile data access", *SIGMOD Record*, vol. 24, n. 4, pp. 10–15, December 1995.

[86] B. D. Noble, M. Satyanarayanan, D. Narayanan, J.E. Tilton, J. Flinn and K.R. Walker, "Agile application-aware adaptation for mobility", *Operating Systems Review*, vol. 31, n. 5, pp. 276–287, December 1997.

[87] Location Working Group, Open Mobile Alliance, http://www.openmobilealliance.org/.

[88] M. Ott, G. Michelitsch, D. Reininger and G. Welling, "An architecture for adaptive QoS and its application to multimedia systems design", *Computer Communications Journal, Special Issue on Quality of Service in Distributed Systems*, vol. 21, n. 4, pp. 334–349, April 1998.

[89] T. Pham, G. Schneider and S. Goose, "Exploiting location-based composite devices to support and facilitate situated ubiquitous computing", in *2nd International Symposium on Handheld and Ubiquitous Computing (HUC '00)*, pp. 143–156, Bristol, UK, September 2000.

[90] G. Pingali, C. Pinhanez, A. Levas, R. Kjeldsen, M. Podlaseck, H. Chen and N. Sukaviriya, "Steerable interfaces for pervasive computing spaces", in *1st IEEE International Conference on Pervasive Computing and Communications (PERCOM '03)*, pp. 315–322, Fort Worth, TX, USA, March 2003.

[91] S. R. Ponnekanti, B. Lee, A. Fox, P. Hanrahan and T. Winograd, "ICrafter: A Service Framework for Ubiquitous Computing Environments", in *3rd Ubiquitous Computing International Conference (UBICOMP '01)*, pp. 56–75, Atlanta, GA, USA, 2001.

[92] S. R. Ponnekanti, B. Johanson, E. Kiciman and A. Fox, "Portability, extensibility and robustness in iROS", in *1st IEEE International Conference on Pervasive Computing and Communications (PERCOM '03)*, pp. 11–19, Fort Worth, TX, USA, March 2003.

[93] Electronics for Imaging, PrintMe™, http://www.efi.com/.

[94] L. McLaughlin III (Ed.), "Line printer daemon protocol", RFC1179, Network Printing Working Group, IETF, August 1990.

[95] R. Fielding, J. Gettys, J. Mogul, H. Frystyk and T. Berners-Lee, "Hypertext transfer protocol, HTTP/1.1", RFC2068, Network Working Group, IETF, January 1997.

[96] R. Droms, "Dynamic host configuration protocol", RFC2131, Network Working Group, IETF, March 1997.

[97] T. Dierks, "The TLS protocol version 1.0", RFC2246, IETF, January 1999.

[98] M. Wahl, T. Howes and S. Kille, "Lightweight Directory Access Protocol (v3)", RFC2251, Network Working Group, IETF, December 1997.

[99] Y. Goland, E. Whitehead, A. Faizi, S. Carter and D. Jensen, "HTTP extensions for distributed authoring: WebDAV", RFC2518, Network Working Group, IETF, February 1999.

[100] E. Guttman, C. Perkins, J. Veizades and M. Day, "Service location protocol, version 2", RFC2608, Network Working Group, IETF, June 1999.

[101] C. Perkins and E. Guttman, "DHCP options for service location protocol", RFC2610, Network Working Group, IETF, June 1999.

[102] W. Zhao, H. Schulzrinne and E. Guttman, "Mesh-enhanced Service Location Protocol (mSLP)", RFC3528, Network Working Group, IETF, April 2003.

[103] M. Román, C. K. Hess, R. Cerqueira, A. Ranganathan, R. H. Campbell and K. Nahrstedt, "Gaia: A middleware infrastructure to enable active spaces", *IEEE Pervasive Computing*, vol. 1, n. 4, pp. 74–83, October-December 2002.

[104] M. Roman, B. Ziebart and R. H. Campbell, "Dynamic application composition: Customizing the behavior of an active space", in *1st IEEE International Conference on Pervasive Computing and Communications (PERCOM '03)*, pp. 169–178, Fort Worth, TX, USA, March 2003.

[105] Salutation Consortium, "Salutation Architecture Specification", http://www.salutation.org/, June 1999.

[106] V. Samar and R. Schemers, "Unified login with pluggable authentication modules (PAM)", OSF-RFC 86.0, Open Software Foundation, October 1995.

[107] G. Sampemane, P. Naldurg and R. H. Campbell, "Access control for active spaces", in *Annual Computer Security Applications Conference (ACSA '02)*, pp. 343–352, Las Vegas, NV, USA, December 2002.

[108] M. Samulowitz, F. Michahelles and C. Linnhoff-Popien, "CAPEUS: An architecture for context-aware selection and execution of services", in *Distributed Applications and Interoperable Systems 2001 (DAIS '01)*, pp. 23–40, Kraków, Poland, September 2001.

[109] R. Sandhu , E. Coyne, H. Feinstein and C. Youman, "Role-based access-control models", *IEEE Computer*, vol. 29, n. 2, pp. 38–47, February 1996.

[110] M. Satyanarayanan, "Caching trust rather than content", *Operating Systems Review*, vol. 34, n. 4, pp. 32–33, October 2000.

[111] M. Satyanarayanan, "Pervasive computing: Vision and challenges", *IEEE Personal Communications*, vol. 8, n. 4, pp. 10–17, August 2001.

[112] M. Satyanarayanan, "The evolution of Coda", *ACM Transactions on Computer Systems*, vol. 20, n. 2, pp. 85–124, May 2002.

[113] B. N. Schilit, N. Adams and R. Want, "Context-aware computing applications", in *Workshop on Mobile Computing Systems and Applications (WMCSA '94)*, pp. 85–90, Santa Cruz, CA, USA, December 1994.

[114] B. N. Schilit, J. Trevor, D. M. Hilbert and T. K. Koh, "m-Links: An infrastructure for very small internet devices", in *7th Annual ACM/IEEE International Conference on Mobile Computing and Networking*, pp. 122–131, Rome, Italy, August 2001.

[115] A. Schmidt, M. Beigl and H. Gellersen, "There is more to context than location", *Elsevier Computers & Graphics Journal*, vol. 23, n. 6, pp. 893–902, December 1999.

[116] J. P. Sousa and D. Garlan, "Aura: An architectural framework for user mobility in ubiquitous computing environments", in *3rd Working IEEE/IFIP Conference on Software Architecture*, pp. 29–43, Montreal, PQ, Canada, August 2002.

[117] A. O. Freier, P. Karlton and P. C. Kocher, "The SSL protocol Version 3.0", Internet draft, work in progress, expires September 1999, Netscape Communications, November 1996.

[118] F. Stajano and R. Anderson, "The resurrecting duckling: Security issues for ad-hoc wireless networks", in *7th International Workshop on Security Protocols*, pp. 172–194, Cambridge, UK, May 1999.

[119] P. Sudame and B. R. Badrinath, "On providing support for protocol adaptation in wireless networks", *Mobile Networks and Applications (MONET), special issue on Wireless Internet and Intranet Access*, vol. 6, n. 1, pp. 43–55, January-February 1999.

[120] Sun Microsystems, "JINI technology architectural overview", White paper, http://www.sun.com/, January 1999.

[121] D. L. Tennenhouse, J. M. Smith, W. D. Sincoskie, D. J. Wetherall, and G. J. Minden, "A survey of active network research", *IEEE Communications Magazine*, vol. 35, n. 1, pp. 80–86, January 1997.

[122] S. M. Thayer and P. Steenkiste, "An Architecture for the Integration of Physical and Informational Spaces", *IEEE Personal and Ubiquitous Computing*, vol. 7, n. 2, pp. 82–90, July 2003.

[123] Organization for the Advancement of Structured Information Standards (OASIS), "UDDI Technical white paper", White paper, http://www.uddi.org/, September 2000.

[124] W3C, "Composite capability/preferences profiles (CC/PP): A user side framework for content negotiation", W3C note, http://www.w3c.org/, July 1999.

[125] W3C, "CC/PP exchange protocol based on HTTP extension framework", W3C note, http://www.w3c.org/, June 1999.

[126] W3C, "HTML 4.0 guidelines for mobile access", W3C note, http://www.w3c.org/, March 1999.

[127] W3C, "Resource description framework (RDF) model and syntax specification", W3C recommendation, http://www.w3c.org/, February 1999.

[128] R. Want, B. N. Schilit, N. I. Adams, R. Gold, K. Petersen, D. Goldberg, J. R. Ellis and M. Weiser, "An overview of the ParcTab ubiquitous computing experiment", *IEEE Personal Communications*, vol. 2, n. 6, pp. 28–43, December 1995.

[129] T. Watson, "Wit: An infrastructure for wireless palmtop computing", Technical Report UW-CSE-94-11-08, Department of Computer Science and Engineering, University of Washington, 1994.

[130] T. Watson, "Application design for wireless computing", in *Workshop on Mobile Computing Systems and Applications (WMCSA '94)*, pp. 91–94. Santa Cruz, CA, USA, December 1994.

[131] IBM, "WebSphere Application Server", http://www.ibm.com/.

[132] M. Weiser, "Some computer science problems in ubiquitous computing", *Communications of the ACM*, vol. 36, n. 7, pp. 75–84, July 1993.

[133] G. Welling and B. R. Badrinath, "A framework for environment aware mobile applications", in *17th International Conference on Distributed Computing Systems (ICDCS '97)*, pp. 384–391, Baltimore, MD, USA, May 1997.

[134] E. Christensen, F. Curbera, G. Meredith and S. Weerawarana, "Web Services Description Language (WSDL) 1.1", W3C note, http://www.w3c.org/, March 2001.

[135] Microsoft Corporation International Business Machines Corporation, "Federation of identities in a Web Services world", White paper, http://www.ibm.com/, July 2003.

[136] M. Potts, I. Sedukhin, H. Kreger and E. Stokes, "Web Service Manageability (WS-Manageability), v1.0", Specification, http://www.ibm.com/, September 2003.

[137] M. Wu and A. Friday, "Integrating privacy enhancing services in ubiquitous computing environments", in *Workshop on Security in Ubiquitous Computing (with UBICOMP '02)*, Göteborg, Sweden, September 2002.

[138] ISO/IEC, "Information technology - Open Systems Interconnection - The Directory: Overview of concepts, models and services", Recommendation X.500, ISO/IEC-9594-1, February 2001.

[139] ISO/IEC, "Information Technology - Open Systems Interconnection - The Directory: Public-key and attribute certificate frameworks", Recommendation X.509, ISO/IEC-9594-8, September 2003.

[140] M, Yarvis, P. Reiher and G. J. Popek, "Conductor: A framework for distributed adaptation", in *7th Workshop on Hot Topics in Operating Systems (HotOS VII)*, pp. 44–51, Rico Rico, AZ, USA, March 1999.

[141] B. Zenel, *A proxy based filtering mechanism for the mobile environment*, PhD thesis, Department of Computer Science, Columbia University, February 1998.

[142] B. Zenel, "A general purpose proxy filtering mechanism applied to the mobile environment", *Wireless Networks*, vol. 5, n. 5, pp. 391–409, September 1999.

[143] P. Zimmermann, *The official PGP User's Guide*, MIT Press, Cambridge, MA, USA, May 1995.