Technical Report CS-2004-35

Enabling Chaotic Ubiquitous Computing[1]

O. Andrei Dragoi and James P. Black
oadragoi@shoshin.uwaterloo.ca, jpblack@uwaterloo.ca

School of Computer Science
University of Waterloo
Waterloo, Ontario, Canada N2L 3G1

Abstract: Today, ubiquitous computing is possible only in handcrafted environments, and not across administrative and technological domains. For this to happen, a middleware "ecology" will need to emerge, so that users can exploit a chaotic environment of competing communication and service providers, and overlapping administrative authorities. Although key elements of the ecology exist today, simple exploitation of unfamiliar (or familiar) ubiquitous environments remains a challenge for users. They need simple tools that open the way to this increasingly rich ecology of devices, services, and information. This paper describes how such tools can be provided through a software abstraction called a frame.

Keywords: ubiquitous computing, software abstractions, services, service access, middleware, web proxy

1. Introduction

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 the rather chaotic environment that results. While most of these elements are a reality, this is not true of the software. This paper proposes a clear software abstraction for ubiquitous computing that can help impose some structure and order on the complex and chaotic environment facing the user.

There are several aspects to this chaos. Ubiquitous computing is challenging because of the variety of players that contribute to the user's experience, and the fact that one cannot reasonably expect all of them to cooperate. Various electronic devices can be used in the environment. Internet Service Providers (ISPs) provide wireless or wired ubiquitous access to the Internet, and multiple ISPs may be available in the same physical area. End-user service providers of various sizes and levels of technical expertise, from a big mall-management company to a small corner store, want to make their services accessible. Some might set up smart spaces (such as presentation rooms), but these spaces may overlap arbitrarily and may not necessarily cooperate since service providers compete for users. For instance, when the user is in a conference room, service providers such as a university, the city, and commercial businesses nearby may

[1] This research was supported by Bell University Laboratories at the University of Waterloo, and by the Natural Sciences and Engineering Research Council of Canada.

have services accessible there. Finally, the user may have a number of simultaneous goals that compete for his or her attention, and that need to be mapped to operations involving electronic services.

In our attempt to bring some simple structure to this chaos, we make a number of assumptions. Users browse the internet with conventional thin clients, mediated by a middleware proxy that can observe and transform web objects passing through it. They achieve their goals by using single tools for simple tasks, and each tool generally applies an operation of some service to data being browsed. Some services may require the user to be in an authenticated role. Users can access a *home node* where they store and retrieve private data.

With these assumptions, our contribution is a software abstraction we call a frame. Each frame implements an operation to be applied to particular data, which could be a web page, an image, a link in a web page, a table in a web page, a paragraph, a word etc. When invoked by the user, a frame must discover and use specific services, and might have to handle localized access credentials for them. Only select, relevant frames are presented to the user, grouped into toolboxes. Frames themselves decide on their relevance to a user, in the context of their use. These tools bring order out of the chaos of ubiquitous services confronting the user. They allow multiple administrative domains and competing service providers to co-exist and even interoperate, have low barriers to entry that can be used to leverage existing electronic services, and enable an ecology of ubiquitous computing in a general setting.

In the next section, we motivate the need for a ubiquitous-computing software abstraction, and describe our model of how the user interacts with the system. Section 3 identifies the functional properties of frames and describes the rationale behind each. Section 4 argues the benefits of this design and describes a proof-of-concept implementation used to validate it. Section 5 further discusses deployment and implementation issues as well as several open problems. Section 6 contrasts our approach with related work. Conclusions follow.

2. Vision and Motivation

Middleware at the edge of the Internet plays a key role in mediating among the various players that interact there: end users, known or unknown to the infrastructure; end-user service providers like stores, hotels, cinemas, travel guides, and print shops; middleware and internet-service providers; and software developers who would provision all or parts of a user-friendly ubiquitous-computing environment.

While the software developers need to recover development expenses, the user needs non-disruptive, functional, and interoperable software. When the user (or the service provider) purchases a ubiquitous-computing space designed for a single purpose such as a meeting room, it is clear to both where and how to add value. In the general case, however, unless clear software abstractions are defined, it is less obvious how value can be added. For instance, what is the model of software licensing and the incentive to write it? How are the sources of individual software components identified? How does one, in fact, identify what brings a specific benefit?

Outside of smart spaces built with a fixed purpose, different software developers design the different “applications.” It is important for a developer to understand where the boundaries of each application are, and to comprehend how the different applications might or should interact. The user of a ubiquitous-computing system is immersed in the physical and electronic environment. For instance, a user might be simultaneously tracking a parcel, finding his or her way through the building, accessing his or her calendar, and then suddenly decide to buy a beverage from a vending machine that happened to be along the way. In other words, the user interacts with multiple “applications” in arbitrary ways. This brings us to the open problem of what an application means, in the special case of ubiquitous computing. It also suggests smaller units of software to simplify development by reducing the complexity for the developer.

Many scenarios discussed in ubiquitous-computing research reflect an assumption that the system should deduce user intentions and fulfill them proactively, with limited user intervention. This approach is unlikely to be successful in a general-purpose system. Moreover, deploying several “parallel” special-purpose systems is likely to be prohibitive. We take the approach that a careful selection of simple tools, presented to the user by the middleware, can provide enough structure and coherence for users to understand and exploit an unfamiliar (or familiar) ubiquitous-computing environment.

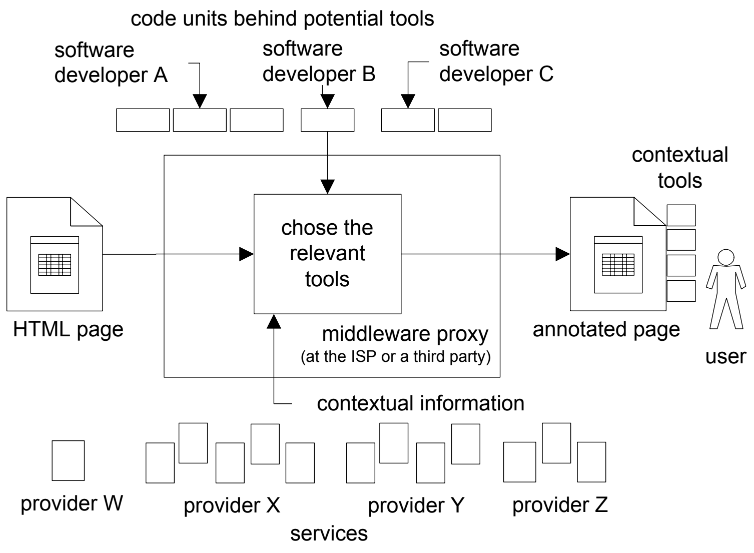

Figure 1. A view of a ubiquitous-computing system

The general-purpose ubiquitous-computing middleware (acting as a proxy) should manage, select, and present only those tools that are currently relevant to the user, and then manage the execution of the software behind the tools (Figure 1). We assume each tool corresponds to a single operation on certain data, operation that might make use of external services. These services may require the identification and authentication of the user in a particular role. We discuss this in more detail elsewhere [8]. In general, we also assume each user has some home node, which is not part of the infrastructure. We expand further on these assumptions below.

Ubiquitous computing is about service access, not about computations on the mobile device. People use mobile devices to browse data, to access services, and to control how these services act on user data [1]. As implied by the figure, we assume browser-based access mediated by a proxy server under the control of an ISP. Competing software developers and service providers want to improve the users' experience by easing and customizing their access to services. Middleware providers deploy an overlay infrastructure to manage and run software that facilitates access to these ubiquitous services. As the user browses, the proxy can modify, annotate, or replace each page requested.

A *service* is any entity that can send, receive, generate, or manipulate data (e.g., and HTML document, an image, a PostScript document, or a text document). Ready examples of services are a printer, a wireless digital camera, a display, or a mobile phone. Services can be offered by software (e.g., banking, translation) or they can be offered by hardware (e.g., printing, storage).

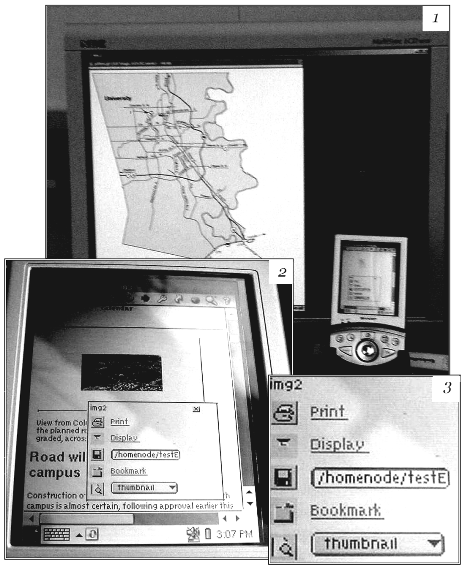

Figure 2. Continuum infrastructure at work

We adopt a simple usage metaphor for the ubiquitous-computing interactions, related to what has been proposed in projects such as Speakeasy [24], m-Links [28], and Satchel [22]. The web page rendered in the browser is augmented with dynamically generated toolboxes of contextual tools that the user can apply to data objects or groups of data objects contained or referenced in the web page. The *data objects* are structured documents (an HTML page, a PowerPoint presentation, a photo), or components of them (a picture in a web page, a table in a web page, a chart in a PowerPoint presentation). Each *tool* in a toolbox corresponds to the user interface of an operation on data. An *operation* can move data between devices and services (e.g. displaying on a nearby projector, acquiring a photo from a digital camera, saving to a disk, beaming to a peer mobile device), or can transform data. It is very specialized, has a minimalist, data-centric interface with the ubiquitous-computing infrastructure, but may be arbitrarily complex in functionality and have arbitrary user interfaces. An operation can use one or more external services. For complex interactions, several operations might be applied by the user in sequence to the same or different data. Some operations, as some service, do not need any input data (e.g. a digital camera), but might generate data objects or have other side effects, so we also define a degenerate case of tools that are associated with no data.

Figure 2 shows an example of interaction implemented in the Continuum prototype (to which we return later in this paper). The user views a map on her PDA. When she finds that the map is not rendered acceptably on the small screen, the user clicks through the map to access a toolbox with tools referring to the map. The toolbox has five tools. She chooses the *Display* tool, to magnify the small map on a nearby public-use display.

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 might or might not have been in a format understood by the display service.

We use the term *publication* to mean the process of dynamically selecting the set of relevant operations to be associated with a particular hotspot, followed by instantiation of the code units that implement each of these operations. Portions of this code are added on the fly to the web pages the user visits to give him or her access to the operations. The code behind these operations is often provided by the middleware provider not by the service provider.

We use the term *decoration* to mean the user-perceived impact of annotating a web page with toolboxes. To what data the tools in a particular toolbox refer, it is a separate issue. We make this association through an UI artefact, called a *hotspot*. For a web page, the hotspot is an icon in the corner of the page; for an image, it is the image itself on which the user can click. If the primary interface with the user is voice, the "hotspot" might be a special keyword pronounced by the user.

A tool starts executing if the user chooses to interact with it. While executing (part at the browser part at the middleware, as we shall see), tools have access to information about services, but also to the values of the attributes associated with users, devices, and services. The attributes and their values are also important parameters to the publication decisions and to tool operation. Examples of attributes are the location of a device (or its user), a certificate asserting that a user is able to act with a certain role (a role certificate), or the URL of the user's home node.

We assume all interactions happen over HTTP, as a *de facto* standard, but any request-reply transport protocol could be used. The user's personal device is assumed to be running a thin client (a standard Internet browser). This data-centric view takes into account that, with all the variability in a ubiquitous-computing environment, the user will relate first to the data he or she needs to access and only second to what can be done to that data and how. Furthermore, we do not assume local availability of stable storage for private data, or that the user carries all the data he or she needs with him.

To perform its function, the middleware has to have access to the unencrypted data the user is browsing (as is often the case of an ISP). In this paper, we assume interactions for which the benefits of convenience overcome the risk of potential security breaches. Target settings are, for example, a mall, a hotel, a university, an airport, or a commercial street. The assumption is that breach of trust produces only nuisance, or at most the loss of small amounts of money. When security is a larger issue, it is assumed that the owners of a service (employers, businesses, governments, law enforcement agencies, the user, etc.) and of the data objects will choose security over convenience, and employ "classic" solutions such as VPNs, direct authentication, and end-to-end encryption.

These assumptions lead us to search for the small, self-contained software abstraction we now call a frame. We expand on frames in the next section.

3. A software abstraction

The developer's task is to implement the software unit behind each tool. We refer to this software unit as a frame. (The term is used to convey the idea of bundling together well-defined functionalities; it has no relation to the HTML *<FRAME>* tag or to link-layer transmission units.) What functionality must a developer include in such a self-contained unit of software that implements an operation? These operations must be developed considering a number of characteristics unique to interactions in ubiquitous computing: an environment rich in electronic devices, provided by a combination of small service providers and hand-crafted smart spaces, multiple administrative domains, arbitrary properties or attributes of services and devices (or their users), and the need for dynamic discovery of both service directories and services. We are primarily concerned with reusable aspects of this design, so we keep the initial description independent of particular programming languages and infrastructural elements. In a subsequent section, we describe our implementation of Continuum, the proof-of-concept prototype we used to test and validate our ideas.

Figure 3 illustrates the functional properties of a frame. The frame logic combines the functionality of the external services, interacts with those services over various protocols, drives the process of authorizing the user to access the service, and drives the user-interface interaction. It can make use of contextual information like attributes and role assertions. The frame logic is typically split between an applet, running in the browser and dealing primarily with the user interface, and a servlet, running in the proxy server. The servlet provides an *onPublication* method to help decide if the frame should be published, responds to user events as reported by the applet (assuming publication occurs), discovers services as necessary, and interacts with them to execute operations. Some frames may not require external services; this is a design issue for the frame developer.

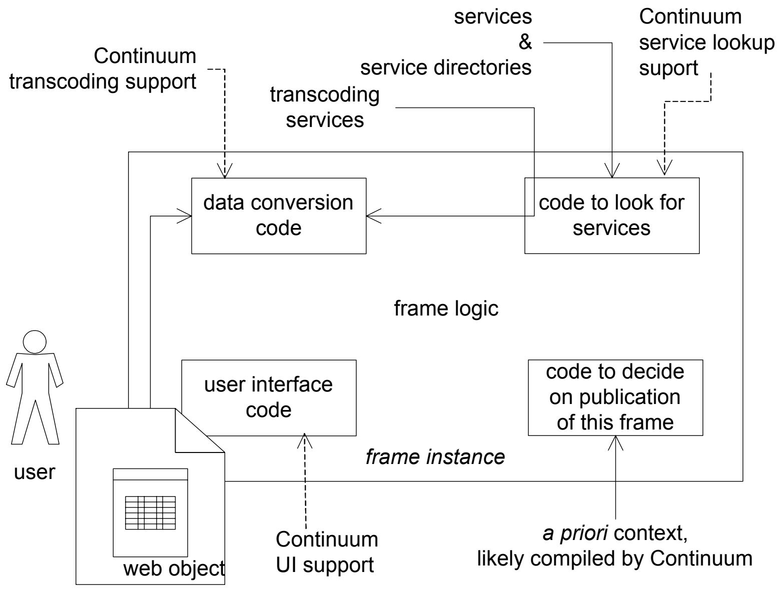
services & service directories
Continuum service lookup suport
Continuum transcoding support
transcoding services
data conversion code
code to look for services
frame logic
user interface code
code to decide on publication of this frame
frame instance
user
web object
Continuum UI support
a priori context, likely compiled by Continuum

Figure 3. Frame functional properties

3.1. User Interface

What must a developer do to implement the frame behind a new tool? A first step is to design and code the user interface (UI).

A tool makes the semantics of the frame apparent to the user, and may gather additional input. Conveying clear semantics to the user can be challenging. Fortunately, users are better than software in dealing with semantics. For example, a button labeled *Start* would break software that expects the label *Play*, but would pose no problems to a user.

In the simplest case, the semantics of a frame can be conveyed through as little as an entry in a menu, like the tools *Print* and *Display* in Figure 2 below. Frames can also have more sophisticated interfaces, such as voice, if the client device supports them.

The underlying assumption is that each frame has a certain UI "real estate" associated with it, allocated during the publishing process, in the toolbox for the hotspot for which the tool has been published. The tools pictured in Figure 1 (on page 3) represent interfaces exposed to the user simultaneously by several frames. There can also be tools associated with no data, all in a special toolbox, accessed through a dedicated hotspot.

The ubiquitous-computing infrastructure could ease the implementation of frame interfaces with special-purpose libraries and templates, through dynamic generation or adaptation of interfaces [16], and through dynamic generation of meaningful labels for services and other URLs [28].

3.2. Data Conversion

The purpose of a frame is to give the user access to specific contextual services. Yet often, even services with identical high-level semantics accept and generate different data types. This is the case even for widely deployed and standardized services such as printers. Some printers accept PostScript while some accept only proprietary languages. To complicate things further, different formats might have the same apparent fidelity to the user (e.g., PostScript and PDF) while others might not (e.g., JPEG and GIF).

In the general case, a frame developer cannot avoid having to control how data-type conversion takes place, because this is tightly tied to the particular semantics of a frame. For example, does *Print* mean printing an outline of the document, does it mean printing what the user sees (potentially a poor quality rendering of the document), or does it mean printing the original?

This notion of data transcoding for the web was first advocated by Brooks *et al.* [4]. Directing type conversions and data versioning 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 the other services it needs to execute its particular operation. Transcoding services that are unavailable can be dealt with like other missing services by not publishing the frame or, if the frame is already published, by revealing the problem to the user.

3.3. Service Discovery

Once the user chooses an operation to execute on an object, that operation must be bound to an actual service. To do this, the frame first obtains references to instances of the services required. Service discovery and use are related. Both depend on the semantics and specific properties of the potential target services. Many current ubiquitous-computing systems are handcrafted, based on a set of simplifying assumptions with respect to the discovery protocols used, the types of services to be encountered, the kinds of activities the user will perform in the setting, and the patterns of interactions that will take place. For example, within the iRoom project at Stanford, Ponnekanti *et al.* [26] report that they felt no need to use existing discovery protocols, because they were served well enough by the event-communication facility at the core of their infrastructure. This results in a system that is highly decoupled in its structure, yet monolithic in application, and is not unlike early attempts in distributed-systems research to hide the distribution, when in fact the user could have taken advantage of it.

In the general case, the semantics of these services and the way they are characterized will still be arbitrary, even if software interfaces of services are simplified to a common, bare minimum (e.g., HTTP, or the minimal interfaces in the Speakeasy project [10]). Furthermore, different administrative domains can choose different service-discovery protocols, with different type hierarchies and conventions. For instance, the SLP [15] template for printers uses the *printer-location* attribute, while in Jini [30], a printer location is expressed as an instance of the class *net.Jini.lookup.entry.Location*. Unlike software, human users can often easily map the semantics of the services encountered into previously known semantics.

The frame will have to commit to a specific set of discovery protocols and, ultimately, to issue the service-discovery query or queries to directories known to the infrastructure. Consider the differences among the semantics of the following three frames: *PrintInTheRoom*, *PrintForDeliveryTomorrow*, and *PrintOnTheDepartmentalPrinter*. All require a printer, potentially the same kind of printer, but there are significant differences in the exact query issued. A frame that is to be robust in discovering and using services with similar semantics but different discovery protocols will have to include significant knowledge of those semantics and protocols. If additional service-discovery protocols need to be supported, the code for the frame may be modified, or an additional frame coded, deployed, and published when appropriate.

By having frames take responsibility for performing discovery, one avoids the need to agree on a discovery protocol, a hierarchy of service types, and the names and semantics of attributes.

3.4. Frame Publication

We have mentioned several times the idea of a frame being published if it is “relevant.” This relevance is subjective because of the difference between what the ubiquitous-computing middleware can know or infer about the user and his or her context, and the actual user intentions, given the situation and the environment as he or she sees it. We conceptualize this difference by using the terms *a priori context* and *electronic context*.

The *a priori* context is a partial reflection of the “real world” in the ubiquitous system. It refers to contextual information about the user and his or her activities and goals, e.g., the user is in a meeting, the user is John. It can come from sensors as in the Context toolkit [7], environment monitors, service directories, inference engines, or even from the user. The corresponding system artifacts of the electronic context are likely to be highly simplified or approximate because of, for instance, the lack of sensors, or sensors of suitable accuracy, the inability to predict the future correctly, or even the lack of a tangible technical definition of certain notions.

Starting from the *a priori* context, the job of a ubiquitous-computing system is to provide contextual means that help the user devise and carry out his or her tasks related to the data object(s) viewed on the mobile device. The set of published frames (or, for the user, the set of tools) represents the electronic context generated by the system.

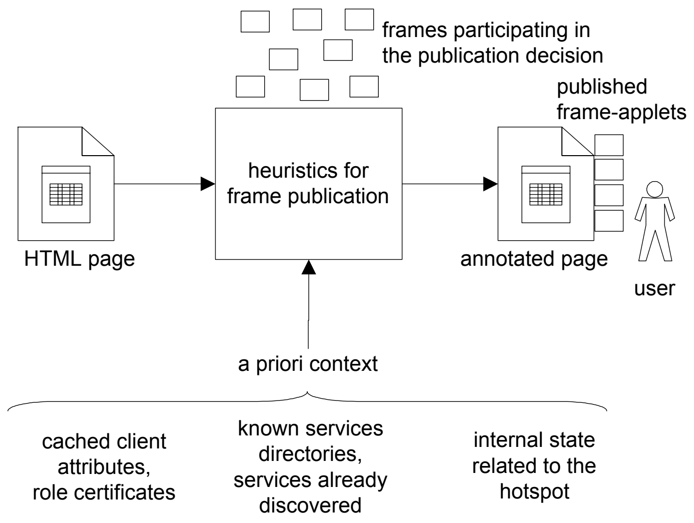
frames participating in the publication decision
published frame-applets
heuristics for frame publication
HTML page
annotated page
user
a priori context
cached client attributes, role certificates
known services directories, services already discovered
internal state related to the hotspot

Figure 4. Frame publication

Figure 4 illustrates this frame publication process and elements of the *a priori* context that can influence it. The relevance and semantics of the *a priori* context elements are tightly linked to the particular objectives implemented by a frame. For example, if the user is a visitor (indicated by a special role assertion) in an office building (indicated by the location attribute of the device), a frame *PrintOnTheDepartmentalPrinter* might not be relevant, while a frame *PrintInTheRoom* could be.

A developer writing a frame could list conditions on the *a priori* context that warrant frame publication. This works for middleware with simple publication heuristics and for simple frames. Otherwise, the only option is to request certain elements of the *a priori* context and to implement a customized frame-publication decision algorithm in the frame. This might include actively discovering certain services or obtaining certain *a priori* context elements directly. Yet, querying service directories can be expensive, so frames might choose publication without first looking for services. On the other hand, this might result in a large number of published frames.

The user should be made aware that the synthesized electronic context can only be an approximation, and he or she should ideally have a means to control the extension of the set of tools proactively. Although not user-friendly, simple examples of such actions are providing URLs to additional "local" service directories, providing information about the user as additional attributes, or even providing the code for the frames of interest.

4. Validating the Frame Concept with the Continuum Prototype

We have refined these general notions by building the Continuum prototype to validate the frame notion, and to get an idea of how complicated frames are to implement, and of what they require in terms of middleware infrastructure. The middleware is mostly implemented in Java. The exercise helped us refine the four functionalities a frame must include, and provided evidence that infrastructure cannot be completely relied upon to implement them on behalf of the frame developer.

In the current prototype, a Continuum middleware server is assumed to run at the last wired hop, controlling frame publication and offering a (server) runtime for frames. From an HTTP perspective, the server runs as an HTTP proxy, but its functionality goes well beyond altering the HTTP traffic. A frame consists of an applet executing at the client (within a browser), and a servlet executing at the Continuum server. Currently, applets are a combination of HTML and JavaScript, to capitalize on the functionality already available in web browsers, and they are mostly responsible for the user interface of the frame. The frame servlets are implemented in Java and can make use of several methods offered by the Continuum runtime.

We have already refer to Figure 2 (on page 4), showing an instance of how Continuum might be used. The *Print* frame in the shown toolbox prints on a nearby printer. Printers are well standardized services, yet, as previously mentioned, their properties vary from one discovery protocol to another. The frame we implemented can discover printers advertised over Jini [30] and SLP [15]. The frame accesses SLP printers with the line printer daemon protocol [23]. Jini objects representing printers are accessed over a specialized Java interface. Currently, we do not use the *javax.print.PrintService* interface, but it would be trivial to do so. The frame handles only PostScript printers; for conversion to this format, it uses methods offered by the Continuum runtime.

Unlike printers, displays are less standardized as network devices (the closest approximation is exposing the display through an X server), in spite of the very standard interface between a display and a computer, and the clear semantics for the users. The *Display* frame in Figure 2 can only locate displays advertised over Jini (because there is no display type defined in SLP) and manipulates the displays through a specialized Java interface. (The frame discovers and uses custom Jini objects that communicate with the remote X display server.)

In general, a frame supplies the specific discovery queries to helper methods provided by the Continuum middleware. The queries can be constructed based on a variety of information available to the frame. The type of hotspot for which the frame will be published, which includes the type of data to which that hotspot associates tools, is an example. Another example is the attributes of the device or the user, such as the current location. Continuum could make use of whatever location technologies are available, and currently takes the very simplistic approach of obtaining the location directly from the user.

The service directories queried are those known by the runtime plus any others known for the particular client. (All the information about a client, such as cached attributes and known service directories, is kept within a soft session.) While some service directories are built into the prototype, Continuum also provides a dynamic mechanism for acquiring relevant service directories. We will only describe it briefly here. More details can be found in a different technical report [8].

Essentially, through formal and informal peer-to-peer interaction with other “local” users, a user acquires roles and knowledge of directories and services where those roles can be exploited. These other local users are affiliated with a local administrative domain or have already acquired knowledge of the local topology. The users exchange role assertions that name a role accepted by contextual services, and indicate directories where such services can be found. These assertions capitalize on both topological knowledge and “the trust knowledge” of the issuing user, and can provide the necessary credentials to use contextual services. The generation and exchange of the role assertions are implemented by specialized frames published to both the issuing user and the recipient. An example is a hotel guest acquiring a role assertion from the reception clerk as part of the registration process. The assertion then gives the guest electronic access to services in and around the hotel.

We also plan to experiment with other mechanisms for discovery and registration of services. Larger service providers can register their service directories with the ISPs covering the area they serve, much like local businesses ensure their advertisements are printed in the local newspapers. Service directories can also be specified in a user’s preferences at the home node, or discovered through mechanisms of specific discovery protocols. For instance, in Jini, multicast can be used to discover service directories on the same network. Research results in direct discovery of web presences for physical services, such as the Cooltown project [20], can also be leveraged.

Returning to Figure 2, the third and fourth frames interact with the user’s home node to store the framed data object or to bookmark it. The home node offers permanent storage accessible over WebDAV [13]. It can be under the control of the user or it can be a purchased service. For access to the home node, the user is challenged for access credentials that are opaque to the middleware. The fifth frame lets the user control the fidelity of the framed image: as an empty rectangle, as a thumbnail, at full size, in full color or grayscale. For this frame, the actual image transcoding is performed by a method supplied by Continuum.

In the prototype, publishing a frame involves only the instantiation of the frame applet, that is, the HTML and JavaScript code that is added to the HTML page. As a matter of design, servlets are stateless, so they can be static classes or instantiated as needed. Any state pertaining to one frame instantiation is kept within the applet. A frame developer provides templates for the HTML and JavaScript code of the applet, in which certain "well-known" macro definitions are available, such as the data-object URL and its MIME-type. He or she also provides a Java class implementing a special-purpose servlet interface. Based on the object MIME-type (if required), the hotspot type, and on other client attributes in the runtime session, the *onPublication()* method of this interface is called to determine if the frame should be published. To create an applet, the various pieces of Java, HTML, and JavaScript code are bundled together by a meta-information XML record, prepared when the frame code is registered with the Continuum infrastructure.

The current publishing algorithm is simplistic. For an intercepted page request, and for each hotspot identified or created in that page, each registered frame is invoked to decide if it is relevant. Frames might need to know if a certain service is available before deciding on their publication, but querying service directories can be expensive due to latency. To alleviate this problem, we plan to implement a cache of services. The frames prototyped so far decide on their relevance without making discovery queries. Those that do need external services, such as *Print* and *Display*, launch discovery queries only if the user chooses to interact with them. If multiple services are found, the user is asked to choose the service he or she wants.

It is important to note that, regardless of who takes the publication decision, stale frames might be published, because of user mobility, of services being stopped, or of optimistically publishing frames that 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 as part of the contract. 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. An user can always opt to force a regeneration of the toolbox. This can be done by reloading the page and hence forcing a rerun the publication algorithm. One can also imagine delivering the toolbox in separate HTML files referenced from the web page, and potentially refreshed more often than the page.

The Continuum prototype offers several simple HTML+JavaScript templates for frame user interfaces. It also offers a limited set of general-purpose JavaScript and Java methods that are available when the applet or servlet executes. We have already mentioned service discovery and transcoding helpers. Other facilities include a standardized communication method between the applet and the server that facilitates caching of client attributes at Continuum, and predefined user interfaces and methods to access the cached attributes from the servlet and the attribute pool from the applet.

5. Prospects and Challenges

Ubiquitous-computing systems are complex. The software abstraction introduced in this paper simplifies the task of the developer. It shifts the burdens of managing software for dealing with this complexity from the user to the developer and the middleware providers.

Several relationships are significant in the resulting system. Users have a business relationship with the ISP. Every time they sign into the ISP's network, they also explicitly or implicitly sign up with the Continuum server. The underlying assumption is that users are aware that the electronic context compiled by the Continuum server is an approximation, and that they might need to seek local information actively about available services and credentials to access them.

Large ISPs seem best positioned to act as middleware providers, through their size, their technical expertise, and the fact they always transmit the packets to and from the mobile clients. The core open issue is how frame code is managed, that is, how Continuum comes into possession of frame descriptions and code. Ubiquitous-computing middleware providers can proactively purchase frame code for popular services in the area they cover, recovering licensing expenses from the flat access rate they charge to users. They might also have business agreements with select service providers, and roaming agreements with other ISPs. Such agreements would include sharing frames to ensure a similar level of service in a foreign network. When the ISP and service providers in the area are more decoupled, one could also imagine services that push frames to the mobiles, code that in the end is cached at the Continuum server for future use by the same or other users.

We see the set of frames at a Continuum server as expanding dynamically and possibly on demand, to suit the preferences and the topology of available services for the area the ISP covers. This brings us to the possibility of having heuristics for publication that can track frame usage history for all users and for each individual user. Alternatively, to reduce the latency to transfer frame code to the client, frames could be cached at the client or pre-fetched, assuming storage is inexpensive. However, the publication decision is still better taken at a well-connected network node like the Continuum server.

In theory, service providers could provide code directly to mobile devices, but it is likely that another level of indirection (the frame), provided by a third party, can better take into account user preference (e.g., localization). Moreover, service providers do not collaborate with each other and might in fact compete, so there is no incentive for them to create software that also makes use of services offered by other service providers. In addition, some service providers are small entities (e.g., the corner store) that would rather minimize the infrastructure they need to deploy. Ideally, they would just have to provide the services.

6. Frames Compared with Other Approaches

Having identified the promising prospects of frames, how do they compare with related notions? The notion of user activity has been proposed as a replacement for the notion of application, in the context of ubiquitous computing, but this notion is too fluid and can change frequently and arbitrarily at the user's whim. In the general case, such changes can rarely be "guessed" from the *a priori* context. Newman *et al.* [24] observe that ubiquitous-computing systems should support the user in assembling available resources to accomplish his or her goals, since it is virtually impossible to have applications for each potential goal. Erickson [12] 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.

Other research tries to design infrastructures that guesses or is provided with the services needed by the current broader task. Then, the infrastructure tries to match the needs with the available services. For instance, in the Aura project [29], longer-lived tasks, such as writing a paper or buying a home, are first-class entities, and the system strives to bind the tasks of a newly-arrived user to suitable services. Continuum takes an alternative stand that such long-term tasks are arbitrarily complex and often cannot be characterized statically. Frames define simpler operations, which are not long-lived, so do not require re-binding, as do Aura tasks.

The notion of a longer-lived task, or rather of a task template, from the Speakeasy recombinant computing project [9] is a less complex notion than in Aura. A task template takes services accepting and generating objects, and coordinates them by specifying the data connections among them. Even a file system can be seen as a service in Speakeasy. Task templates can be predefined, or the user can assemble them manually. In Speakeasy, services are assumed to have minimal, "recombinant interfaces," for data transfer, metadata, and control. Speakeasy is close to Continuum in its endeavors. From a system perspective, frames are more active than a task template, containing 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 in conjunction with a certain data object. From a user perspective, a frame does not perform the operation by itself. Instead, it is associated with an individual data object on which one can perform an operation, advancing the user in his or her goal. Conceivably, to achieve the same results as certain Speakeasy tasks, a user might have to interact with several Continuum frames in sequence.

Be they long-lived tasks, frames, or even full-fledged applications, all these software artefacts represent another level of indirection in the user's interactions with services. A frame offers access to a set of existing services in the form of operations to be applied to data viewed by the user. The Appliance Data Services (ADS) project [17] looks at the complementary problem of how to apply device and/or context-dependent services to data generated by the user's simple and single-purpose mobile devices.

From a user perspective, the idea of the frame interface (the tool) is very similar to the notion of tools, task panels, smart tags, and the context menus in office- productivity suites on conventional computers. However, from a system perspective, there are several fundamental differences in the requirements and the way frames are created, published and executed. First, external services are needed to apply an operation, and these services are not under the control of the designer of the software unit (the frame) that sits behind each tool. These services need to be found, but one cannot have built-in references to them, and cannot rely on a well-known registry for bootstrapping. The publication of a given frame is usually determined by the services available at the user's location. The user's current activity is not as well defined as in dedicated applications like a word processor. Furthermore, frames are published and execute in a heterogeneous environment, with different administrative domains.

Compared with traditional software units, frames can be as simple as proxies, but usually, unlike proxies, they will add additional functionality and find the services they need actively. Unlike drivers, frames interface directly with the user, not with an operating system. Web Services [3] are on par with other software services used by frames. Unlike frames, which manifestly convey certain semantics directly to the user, Web Services have a service contract with other pieces of software, and commit to a message structure rather than to certain functionality. Similarly, DCOM objects and JavaBeans commit to interfaces with other software components.

The design of individual Continuum frames can benefit from research into the adaptability of mobile applications, where one can identify several models: data-centric adaptation, such as the MPEG video player for the Odyssey platform [25], protocol-centric adaptation as in the use of protocol filters [32], application-centric adaptation such as the speech-recognition application for Odyssey [25], and hybrid adaptation (a combination).

Systems research projects in ubiquitous computing, like Microsoft's EasyLiving [5] and Stanford's iRoom [18], often concentrate on well-equipped smart spaces designed to accommodate specific activities such as a presentation, known at system-design time. One of the problems with current ubiquitous-computing solutions is the lack of generality and interoperability. The approach considered in this paper facilitates interoperability of existing or proposed solutions without requiring changes to deployed services. By not assuming pre-existing relationships between the user and the service providers, it facilitates access to services offered by small service providers. It also enables interactions with contextual services in ways not foreseen at system-design time.

Today, ubiquitous overlay solutions offer access only to a few types of service, and they are hardly transparent for the user. For instance, PrintMe Networks^{TM} [11] is a product marketed to hotels and mobile users. It facilitates printing from a mobile device to a printer in a hotel, printing to a nearby affiliated copy center, and printing with in-room or overnight delivery. A service provider (a hotel) installs additional hardware provided by the middleware company. The user also has to agree to use the same middleware company, and he or she has to manually enter the coded URL for the particular location where he or she wants to print. There are no mechanisms to offer access to an additional service type. This likely implies adding hardware at the service provider, possibly establishing a business relationship with another middleware company, and communicating URLs to users for the services. None of this is likely to go smoothly.

The design of ubiquitous-computing systems is an open area of research. Kindberg *et al.* give several design principles for ubiquitous-computing systems [21]. From an infrastructure-deployment perspective, one approach in related work is to design the system from ground up, usually on top of Java, such as with the one.world architecture [14]. Other projects require an overlay infrastructure to support mobile software units that devices themselves inject into the network. This is the case of the context-aware packets in Capeus [27], and the device-associated mobile agents in the Adaptive Personal Mobile Communications Architecture [19]. As Continuum, they draw from active-network research [31] and mobile agents [6]. However, Continuum frames, or rather the frame applets, are a very simplified case of mobile code in the sense that they are added dynamically to web pages in transit to the user.

The Continuum infrastructure for publishing frames could well be built on top of Open Pluggable Edge Services (OPES) [2]. This is a recent standardization effort towards a framework for invoking, authorizing, and deploying distributed application services such as content transcoding and adaptation, content assembling, logging and accounting. In that respect, it is orthogonal with the framework described in this paper, which focuses on how to help the user access services, including ones built on, or benefiting from a deployed OPES infrastructure.

7. Conclusions

Given the characteristics of a ubiquitous-computing environment, we identify the functional properties that a software building block for ubiquitous-computing must have. A corresponding software abstraction is proposed, the Continuum frame. We show how this design was validated and refined through a proof-of-concept prototype.

Frames provide a simple software abstraction that can enable general, chaotic ubiquitous-computing ecologies with niches for users, ISPs, end-user service providers, and software developers. Middleware proxies, operated by one or more ISPs, provide a hosting environment where simple tools for service access can be deployed, the tools and the services can be discovered by users, and user goals are achieved through sequences of simple operations facilitated by these tools. Security and the competing interests of service providers and users can be accommodated, and various commercial entities can provide software and services for profit, with low barriers to entry into the ubiquitous-computing ecology. Software developers can develop and market small tools and/or related toolsets, and users require neither special software on their devices nor advance knowledge of the contexts in which they find themselves. Each service (or even each frame) can implement its own models for authentication and revenue generation. The proxy providers can charge users for access to the services, and/or can charge other providers for making their services available in the proxy. Users may or may not need an existing business relationship with the proxy provider or with the other service providers. These questions can be decided by market pressures and the values added by the tools and services.

8. References

[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] 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.

[4] 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.

[5] 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.

[6] 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.

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

[8] O. A. Dragoi and J. P. Black, “Discovering services is not enough”, Technical Report CS-2004-36, School of Computer Science, University of Waterloo, August 2004.

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

[10] 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.

[11] Electronics for Imaging, "PrintMeTM", http://www.efi.com/.

[12] 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.

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

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

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

[16] 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.

[17] 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.

[18] 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.

[19] 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.

[20] 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.

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

[22] 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.

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

[24] 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.

[25] 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.

[26] 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.

[27] 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.

[28] 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.

[29] 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.

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

[31] 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.

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