# Applying Dynamic Integration as a Software Infrastructure for Context-Aware Computing

**Gregory D. Abowd & Anind Dey**
College of Computing & GVU Center
Georgia Institute of Technology
Atlanta, GA 30332-0280
USA
{abowd,anind}@cc.gatech.edu

**Andy M. Wood**
School of Computer Science
University of Birmingham
Edgbaston, Birmingham, B15 2TT
UK
amw@cs.bham.ac.uk

**ABSTRACT**

Much of the software engineering literature examines techniques and practices that help us to build systems that we have been building for many years already. While there is merit in seeking ways to raise the floor of software engineering practice, we also have an obligation to research software design issues that push the envelope of existing computing technology. One of the features of future interactive computing environments is that they will provide context-aware services that leverage off of knowledge of a person's physical state and surrounding environment. With the proliferation of network-based computing services that are a characteristic of an emerging ubiquitous computing society, there is a real issue with providing a software infrastructure that will support context-aware services. In this paper, we examine software engineering work on component integration and introduce a prototype infrastructure that provides a dynamic and scalable context-aware computing environment. We will describe how we have applied this component infrastructure to build a futuristic personal information management system involving automatic cooperation between desktop, network and mobile services.

**KEYWORDS**

ubiquitous computing, mobile computing, context-aware computing, software integration, personal information management

## 1 INTRODUCTION

Software engineering research has become a lagging indicator for software development, particularly in the area of interactive software. This was not always the case. In the late 60's and early 70's, the seminal work on abstract data types that lead to object oriented design and programming appeared at the same time that the initial graphical user interfaces (GUIs) were being developed. The natural link between object hierarchies and widgets in a GUI-building toolkit was discovered long before GUI environments were commonplace. Contrast the software engineering research of the 70's with that in the 90's. The majority of the software engineering literature examines how to apply tools and techniques to systems and technology that have been prevalent for at least 10 years. This is a disturbing trend.

There is, of course, merit in today's dominant mode of research, but we need to focus some software engineering research effort on more futuristic applications. By doing so, the software engineering community can again become a leading indicator for software systems that will be deployed in the coming ten years. Without this shift in perspective, the software engineering community is sending out the wrong message —that our research is not intended to shape the landscape of future computing and is therefore of no interest to those who aspire to invent tomorrow's paradigms of interaction. Software engineering needs to become once again a leading indicator for the rapid pace of technology change.

Interest in *ubiquitous computing* has risen over the past few years [25, 26, 3] and one of the emerging research themes is *context-aware computing* [1]. In a computing environment with universal access to information anywhere and at any time, the end user will need leverage to help tame the deluge of technology. There is a lot of information surrounding the end user —the user context— that can be sensed and used to predict the kinds of information needed and the form in which that information should be delivered. From a software perspective, context-aware computing demands an infrastructure to allow intelligent mediation between software components, allowing them to act together in ways that might not have been predicted by the original designers [27].

The software engineering technique that shows much promise for context-aware computing is dynamic component integration. The remainder of this paper will demonstrate how dynamic integration techniques can support context-aware computing for future interactive environments.

**Overview of paper**
In Section 2, we will further outline our particular focus in the area of ubiquitous computing, context-aware computing and discuss the application to personal information management. In Section 3, we review context-aware computing and software integration mechanisms. In Sections 4 and 5, we will describe a generic dynamic integration infrastructure and show how it was used to provide the mechanisms for a novel environment for personal information management, called CyberDesk. We discuss a future extension of the integration framework to better support mobility of people and devices in Section 6. We conclude in Section 7 with a summary of the major software engineering contributions our research has had on context-aware computing.

## 2 UBIQUITOUS AND CONTEXT-AWARE COMPUTING

The history of computing is filled with examples of radical paradigm shifts in the way humans interact with and perceive technology. The vision of ubiquitous computing —first expressed by Weiser [25] and grounded in experimental work done at Xerox PARC— holds the promise of yet another paradigm shift. The defining characteristic of ubiquitous computing is the attempt to break away from the traditional desktop interaction paradigm and move computational power into the network and environment that surrounds the user. Rather than force the user to search out and find the computer's interface, ubiquitous computing suggests that the interface itself can take on the responsibility of locating and serving the user.

Weiser admits that it is the applications themselves that make ubiquitous computing a viable research topic for computer science (and other disciplines)[26]. With that in mind, our research in ubiquitous computing has been strongly influenced by the applications which we have chosen to explore. One application domain we have investigated is personal information management. Today, there is a growing number of personal devices and applications, on and off the desktop, that allow us to keep track of our own personal repository of electronic information. Currently this information includes contact information, schedules, e-mail communications, but will extend to encompass a much greater portion of our everyday lives.

As users begin to rely more and more on electronic information, they will expect it to be available to them in a variety of different situations —while in their office, at home, on the road. With the proliferation of mobile devices, it is possible to have access to personal information anywhere, but this is currently done at the expense of having to replicate similar information on a variety of devices. The promise of reliable, ubiquitous networking services should relieve the user of the bother of explicit replication or synchronization of data to the point where it is no longer a concern where information is located.

Universally accessible data is only part of the challenge, however. The relationship between data is important to the user, and very difficult to track as it becomes easier to acquire electronic information. Knowledge of the user's *context* —what piece of information they are currently attending to, where they are located when they look at some information, the time of day, or the people around them— can help to predict when relevant services might best be presented to a user. This requires a software infrastructure that can detect contextual information and then use it to offer advice to the user. It is this latter advice-giving feature that is the focus of this paper.

### 2.1 A Scenario

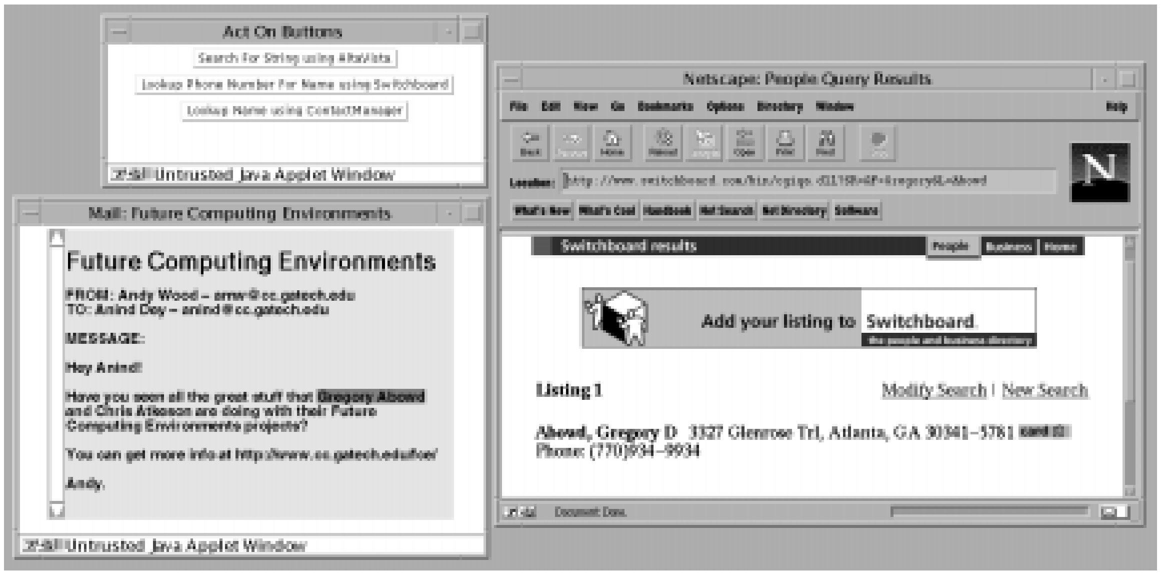
We illustrate the kind of flexible behavior we are aiming to support through a simple scenario using a system we have built, called CyberDesk. Further information on CyberDesk is provided later in this paper and other publications [28, 8].[1] The following scenario is illustrated in Figure 1. As seen in the figure, a user is checking e-mail, and reads a message from a friend about some interesting research. The user decides to find out more and highlights the name of the person mentioned in the message. The interface provides a separate window of actions that can be acted upon based on the name:

- search for the selected text in the Web-based service AltaVista;
- search for a phone number and mailing address using the Web-based service Switchboard; or
- lookup the name in a desktop-based contact manager.

The user wants to contact this researcher, so he checks to see if the name is in the contact manager first. It isn't, so he selects the Switchboard option and retrieves the desired information.

We emphasize some important features of this simple scenario. The services being accessed can reside anywhere —on the user's desktop machine, on the Internet, or even on a mobile device such as a personal digital assistant (PDA) that is connected via wireless network [17]. Also, the user does not need to know what services

[1]We are limited in this paper to describing only a few possible scenarios. To aid the reader, we have provided a Web-accessible location for experiencing more of the behavior of CyberDesk. Go to http://www.cc.gatech.edu/fce/cyberdesk.


Act On Buttons
Search For String using AltaVista
Lookup Phone Number For Name using Switchboard
Lookup Name using ContactManager
Untrusted Java Applet Window
Mail: Future Computing Environments
Untrusted Java Applet Window
Netscape: People Query Results
File Edit View Go Bookmarks Options Directory Window Help
Back Forward Home Reload Images Open Print Find Stop
What's New What's Cool Handbook Net Search Net Directory Software
Switchboard results
People Business Home
Add your listing to Switchboard
Modify Search | New Search
Document: Done.

Future Computing Environments

FROM: Andy Wood – amw@cc.gatech.edu
TO: Anind Dey – anind@cc.gatech.edu

MESSAGE:

Hey Anind!

Have you seen all the great stuff that Gregory Abowd and Chris Atkeson are doing with their Future Computing Environments projects?

You can get more info at http://www.cc.gatech.edu/fce/

Andy.

Location: http://www.switchboard.com/bin/cgiqp.dll?SR+4F+GregoryA+Abowd

Listing 1

Abowd, Gregory D 3327 Glenrose Trl, Atlanta, GA 30341–5781
Phone: (770)934–9934

Figure 1: An illustration of a typical scenario using CyberDesk to automatically integrate desktop, network and mobile data services.

are available, as relevant services are suggested automatically by the CyberDesk infrastructure and made available to the user based on the current context. The current context in this scenario is indicated explicitly by the user based on text that has been highlighted with a mouse, but we can also have more implicit context such as a user's position, trigger integrating suggestions.

## 2.2 Context-awareness and Software Integration

Two issues arise when considering automatic integration of personal information services, and both have software engineering ramifications. First, we have to provide ways to predict when the integration should be offered to the user. This is the crux of the context-aware problem in this application domain. In the scenario above, the user simply indicates a piece of information in a message and the system then infers how that information can be used by other available services. We need to provide a flexible context inferencing engine that can work on many different types of information, such as strings displayed on a screen as described in our scenario, position information for a mobile user, current time, knowledge of people around a given user, and even knowledge of the physiological state of a user.

A second issue is to provide an infrastructure for integrating software applications. Software applications often work on similar information types such as names, addresses, dates, and locations. Collections of applications are often designed to take advantage of the potential for integration via shared information. As an example, an electronic mail reader can be enhanced to automatically recognize Web addresses, allowing a reader to select a URL to automatically launch a Web browser on that location. Even more complex and useful integrating behavior is available in a number of commercial suites of applications (e.g. Microsoft Office 97, Lotus SmartSuite, WordPerfect Suite).

There are some limitations, however, to the current approaches for providing this integration that impact both the programmer and the user. From the programmer's perspective, the integrating behavior between applications is static. That is, the behavior must be identified and supported when the applications are built. The programmer has the impossible task of predicting all of the possible ways users will want a given application to work with all other applications. What results is a limited number of software applications that are made available in an integration suite.

From the user's perspective, integrating behavior is limited to the applications that are bound to the particular suite being used. Further integration is either impossible to obtain or must be implemented by the user (e.g., by cutting and pasting between application windows or by end-user macro programming). In addition, the integrating behavior has a strong dependence on the individual applications in the suite. If a user would like to substitute a comparable application for one in the suite (e.g. use a different contact manager, or word processor), she does so at the risk of losing all integrating behavior.

Given these software engineering considerations, our goal is to provide a more flexible framework for integrating software behavior based on knowledge of a user's context. We want our solution to work under the assumption of a networked and heterogeneous operating environment. We aim to reduce the programming burden in identifying and defining integrating behavior, while at the same time retaining as much user freedom in determining how integration is to occur.

## 3 BACKGROUND

Before we describe our infrastructure for context-aware software integration, we will provide an overview of context-aware computing and software integration techniques. This review will help to place our work properly in the research areas of ubiquitous computing and software engineering.

### 3.1 Context-Aware Computing

We define context-aware computing generally as work that leads to the automation of a software system based on knowledge of a user's physical, social, emotional or informational state. Probably the most successful application of context-aware computing has been automated help systems that use knowledge of a user's informational state and history of interaction to provide assistance with complex software programs. A good commercial example of this is the Microsoft Office Assistant. The user interface community also has a sub-area of research called adaptive systems, which typically involves building a model of user behavior that can be codified and used as a building block in a software system to provide services such as help or self-adapting menus [21]. Computer vision researchers have used computational perception techniques in an attempt to match actual facial expressions with some prescribed expressions indicating the state of the human (e.g., smiling, frowning, surprised, etc.) [9]. Though this work does not claim to be a way to predict human emotions, there is a clear suggestion of how this and related perception research can improve the quality of contextual information that can be gathered. Picard's work on affective computing [16] suggests a similar objective, only through the use bio-electric signals, coupled with theories on emotion and cognition.

A significant body of work in mobile computing takes advantage of the most significantly changing context of a mobile user —location [2, 12, 24, 6]. The initial ubiquitous computing research at PARC provided location-aware services for a handheld device called the PARCTab [24], and resulted in a generalized programming framework for describing location-aware objects [20]. Using informational context, such as what is shown on a user's graphical display (as depicted in the scenario of Section 2.1) has also been the subject of work done at Apple [5] and Intel [15]. This work is the most closely related work to our own and we will discuss it further in the next section on software integration.

### 3.2 Software Integration

The general topic of software integration is well researched. We will focus on those aspects of integration that are relevant to the kind of infrastructure we require for context-aware, self-integrating software.

Our underlying infrastructure allows dynamic integration of isolated services at run-time. Such mediation consists of two basic steps: registration of components and handling of events. This provides for the kind of flexible coordination or mediation between different components. We can compare our integration infrastructure with some other well-known systems, such as UNIX pipes, Field [18, 19], Smalltalk-80 MVC [11], Common Lisp Object System (CLOS) [7]. UNIX pipes act as mediators that integrate UNIX programs. They are limited to reading and writing streams of data, stream outputs can only be input to one stream, and they use only a single event. Field (and its extension Forest) integrate UNIX applications that have events and methods which can be manipulated through a method interface. Similar to our work, Field uses centralized mediation and implicit registration, allowing greater runtime flexibility. However, it suffers from the use of special object components, creating inconsistencies. Smalltalk uses a general event mechanism like CyberDesk, but it merges relationships between components into the components themselves, limiting flexibility. CLOS uses wrappers to access data and methods within objects, as we do, but it limits the action a component can perform to a simple method call and return, thereby limiting its usefulness. Sullivan and Notkin [22, 23] have developed a very flexible dynamic mediation system. However, their system allows only one-to-one relationships between components and requires explicit registration of event-action pairs.

We depend on the use of component software accessible across a network connection, similar to CORBA (Common Object Request Broker Architecture) [14], Microsoft's Common Object Model (COM) and Object Linking and Embedding (OLE) [13]. OpenStep [4] and others. A main distinction in our work is the requirement for a dynamic registry that records the presence of interacting components. At a higher semantic level, the agent research community has also spawned efforts to provide for integration of large-scale software systems [10]. Such efforts have been sponsored by the DARPA Knowledge Sharing Effort and have produced specification languages such as the Knowledge Query and Manipulation Language (KQML) and the Knowledge Interchange Format (KIF).

There are two systems in particular that provide functionality in the domain of personal information management similar to the scenario described in Section 2.1. They are Intel's Selection Recognition Agent [15] and Apple Research Lab's Data Detectors [5].

Intel's Selection Recognition Agent uses a fixed data type-action pair, allowing for only a static set of actions for each data type recognized. The actions performed by the agent are limited to launching an application. When a user selects data in an application, the agent attempts to convert the data to a particular type, and displays an icon representative of that type (e.g. a phone icon for a phone number). The user can view the available option by right-clicking on the icon with a mouse. For applications that do not "reveal" the data selected to the agent, the user must copy the selected data to an application that will reveal it.

Apple Data Detectors is another component architecture that supports automatic integration of tools. It works at the operating system level, using the selection mechanism and Apple Events that most Apple applications support. It allows the selection of a large area of text and recognizes all user-registered data types in that selection. Users view suggested actions in a pop-up menu by pressing a modifier key and the mouse button. It supports an arbitrary number of actions for each data type. When a data type is chosen, a service can collect related information and use it, but this collected information is not made available to other services.

The main limitations of the Intel and Apple work for context-aware computing is the inability to accept a very rich set of context input. These systems report support for only displayed information. We aim to support other forms of context such as time and position. Extensions to our initial infrastructure to support chaining and combining also provide much more powerful context-inferencing capabilities, as we will discuss.

## 4 APPLYING AN INTEGRATION FRAMEWORK

In this section, we describe a Java-based implementation of the dynamic integration infrastructure and its application for a futuristic personal information management system as described by the scenario in Section 2.1.

The starting point of our Java implementation was an infrastructure called CAMEO, a C++ toolkit developed by Andy Wood [29].[2] The CAMEO infrastructure defines a component-based framework in which individual components can observe the activities of other components and manipulate their interfaces. A centralized service allows for dynamic registration of components and runtime support for querying the interfaces of registered components. Observation and manipulation of other components and the dynamic registry services of CAMEO were sufficient motivation for us to port to Java and take advantage of simpler cross-platform network access to a multitude of Web-based, mobile and desktop information services.

### 4.1 The Architecture of CyberDesk

The CyberDesk system consists of five main components: a Registry, information services, type converters, an Integrator, and a user interface. The Registry maintains a list of components in the system and the interfaces that each supports. The information services are the tools and functions the user ultimately wants to use, such as an e-mail reader, a contact manager, or a Web-based search engine. These services register their interfaces with the Registry and announce events that provide data/information to the rest of the system (e.g., the name selected in the e-mail message in the scenario). The type converters accept announced data from the system and convert it recursively to other forms of data that may be useful (e.g. a string being converted to a URL). The Integrator uses the Registry to automatically find matches between user data and the services that can use the data, a task that would normally be performed by the system designer. The matched services are then displayed to the user through the user interface for integration.

The run-time relationship between the components (not including the Registry) is depicted in Figure 2. The components are described in greater detail below.

### 4.2 Registry

The Registry keeps a directory of all the other components in the system, what interfaces they can support, and what data they can provide, if any. As each component joins the CyberDesk system, it provides this information to the Registry. Some components, upon registering, tell the Registry that they are interested in other components. Hereafter, whenever a component joins or leaves the system, the Registry notifies these interested components. The Registry also provides both a white pages and yellow pages service. When queried, the Registry provides information (reference and interfaces) on individual components. It can also provide a list of components that support a particular interface.

### 4.3 Services

Services are end-user function calls that perform actions on provided data. Services can be stand-alone or part of a larger application. Examples of stand-alone services are network-based Web CGI scripts such as finding a phone number and address using Switchboard or search-

[2] For further information on CAMEO, see http://www.cs-.bham.ac.uk/ amw/cameo.

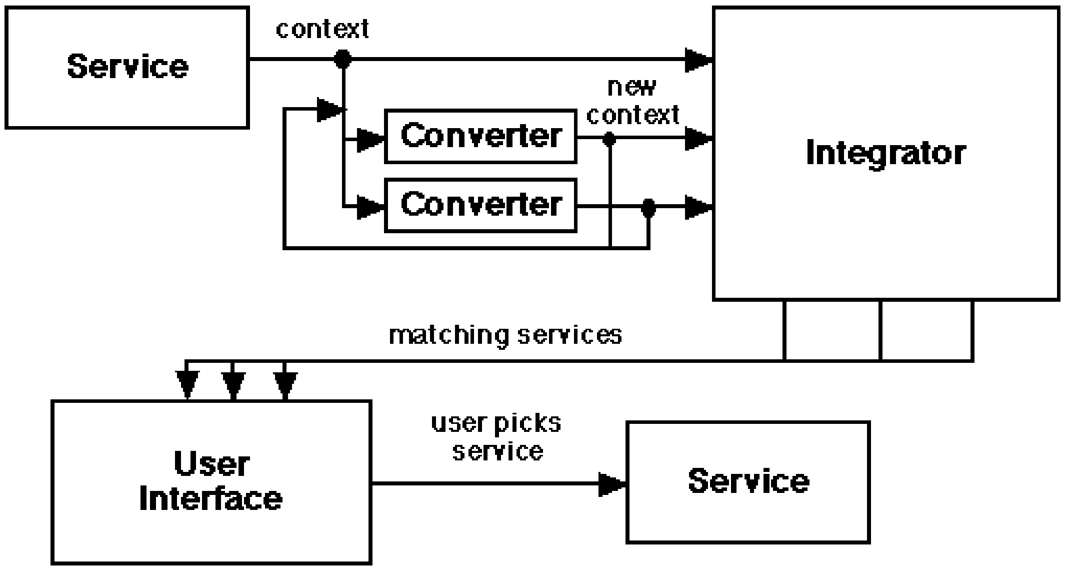


Figure 2: The run-time architecture of CyberDesk. Arrows indicate the flow of information and control in the system.

ing for some text in AltaVista. Examples of services in larger applications are creating an entry or searching for a name in a contact manager, or loading a schedule in a day planner.

Service wrappers are used to integrate existing services into CyberDesk. These wrappers adapt the interfaces of the existing services to conform with the CyberDesk registration and communication requirements. They make the functionality of the services accessible to other components, and provide methods for communicating with other components and registering their interfaces with the Registry.

Services not only provide functionality to the user, but they can also provide data to the system, as seen in the simple scenario. When users select data with a mouse in an application, that data is observed by interested components (a subsect of the type converters and the Integrator described below). The author of the service wrapper determines what information and functionality is made available to the CyberDesk system.

This announced data is the contextual information, as its origin is some user activity, such as selecting text with the mouse.

### 4.4 Type Converters

Type converters are components that take data in the system and attempt to convert it to other forms of data. They use simple techniques to provide complex and intelligent-like behaviors to the system. The example in the scenario showed converting from a string to a name. Data in the system can come from other actions than selection with a mouse. For example, position services provide location information such as coordinates within a space. Type converters can be used to convert these coordinates to a room within a building. This allows the user to see services that are only available within a particular room. Type converters create additional data types to match services against (e.g. a name was converted from a string in the scenario).

The type converters provide a separable context-inferencing engine with arbitrary power. As the conversion abilities of the converters improves, the ability of the system to make relevant service suggestions also improves. Therefore, the apparent intelligence of CyberDesk is also contained within the type converters. Since the type converters are represented as a collection of Java classes, it is a simple matter to boost the overall power of this context inference engine without impacting any of the functionality of the rest of the system.

As mentioned in the section on the Registry, some components are interested in the addition and removal of other components. Type converters are examples of this. Type converters monitor which services are added and removed from the system, so they can determine which components can provide data, and observe those components.

### 4.5 Integrator

The Integrator also observes components that can provide data. It uses this information to find services that can act on the data. For example, when the user selected a name in the e-mail message, both a string and a name (via a type converter) were made available to the system. The Integrator took that data and found services that could act on both strings and names.

When components register or remove themselves from the Registry, the Integrator is notified. The Integrator uses this information to update its list of components to observe data from and to update its list of components that can act on various types of data. For example, when the Switchboard service is added to CyberDesk, it registers that it can perform a function on name information. The Registry notifies all components interested in the addition and removal of components: type converters and the Integrator. The Integrator contacts the Registry to determine the kind of interface the Switchboard service supports and finds out that it can act on name data. When name data enters the system, the Integrator makes the Switchboard service available to the user.

### 4.6 User Interface

When the Integrator finds matching services for the data it has observed, it makes these services available to the user. We have experimented with creating buttons on a separate window to display the suggested services to the user, as shown in Figure 1. Each button is associated with a service and the data the service can act on. When a user clicks on a button, the service is executed with this data.

The user interface, like the other components, is completely interchangeable. If the provided user interface does not meet with the user's approval, it can be easily replaced by another user interface that better informs the user of the connection between his current context and suggestions for future actions based on that context.

### 4.7 Scaling up CyberDesk

These components make up the CyberDesk framework, supporting the automatic, context-aware integration of software applications. We have tested out the scalability of CyberDesk by adding more and more services and context types. Standard desktop applications currently included in CyberDesk prototype include two e-mail browsers, a calendar, a scheduler, a contact manager, and a notepad. Other services available in CyberDesk include over 70 Web-based services, a recognition system, and a positioning service. CyberDesk currently uses 10 different data types (and associated type converters): strings, dates, times, phone numbers, names, e-mail addresses, mailing addresses, rooms, position coordinates, and food.

For ease of experimentation, the initial CyberDesk prototype ran within the environment of a Web browser and we soon realized a problem with scale. Each service supported by CyberDesk is presented to the system as a Java applet. Every service is available to the system at all times, meaning that a potentially large number of applets would be resident in the browser's Java virtual machine. Current browsers are rather limited in the number of applets that can reside concurrently. To alleviate this bottleneck, we have reimplemented CyberDesk independent of the browser environment.

## 5 EXTENSIONS TO CYBERDESK

The initial CyberDesk framework described above is quite adequate for experimenting with context-aware personal information management. It is also somewhat similar in functionality with the Intel and Apple work described in Section 3. In this section, we describe a number of extensions to the initial framework.

### 5.1 Chaining Integration Suggestions

A shortcoming with the initial system is that all context must be directly derivable from observed context information explicitly generated by the user. In order to get to services related to other data types, the user has to explicitly work with those data types. For example, a user selects a name in the contact manager and would like to send an e-mail to this person, but does not have the person's e-mail address. With the initial system, the user must select the name, choose a service that looks up an e-mail address for a given name, enter that e-mail address into the CyberDesk system (add it to the contact manager, for example), and then choose a service that allows him to send an e-mail message using that e-mail address. This is more complex than it needs to be.

A simple extension, *chaining*, decreases the level of complexity necessary to perform such actions. Chaining is the process of generating additional context for the purpose of increasing integrating behavior. Many services take one type of data and return another form of data through a graphical interface, such as a Web browser. Examples of these services are e-mail address lookups, phone number lookups, and mailing address lookups. By making simple modifications to the service wrappers, services can be made to behave like type converters, taking one form of context and returning another. The modifications to the CyberDesk system included parsing the data encoded in the graphical interface to obtain the new data and supplementing the registration information of services to be more like that of type converters. Now services can suggest actions directly

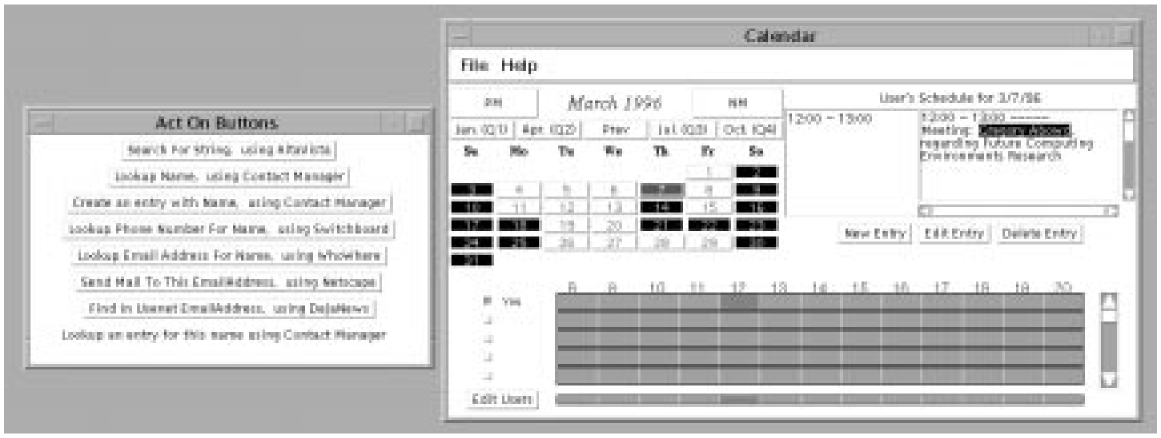

Act On Buttons
Search For String, using AltaVista
Lookup Name, using Contact Manager
Create an entry with Name, using Contact Manager
Lookup Phone Number For Name, using Switchboard
Lookup Email Address For Name, using WhoWhere
Send Mail To This EmailAddress, using Netscape
Find in Usenet EmailAddress, using DejaNews
Lookup an entry for this name using Contact Manager
Calendar
File Help
March 1996
Prev
Jan. (Q1) Apr. (Q2) Jul. (Q3) Oct. (Q4)
Su Mo Tu We Th Fr Sa
User's Schedule for 3/7/96
12:00 - 13:00
Meeting:
regarding Future Computing Environments Research
New Entry
Edit Entry
Delete Entry
Edit Users

Figure 3: An example of the chaining extension to CyberDesk.

related to the user's context and actions indirectly related to the user's context, reducing the effort required by the user to find these services.

As an example, assume a user is reading an appointment in her scheduler and selects the name of the person she is supposed to be meeting (see Figure 3. As an experienced user, she expects to be presented with a list of all possible services that can use a name: search for a phone number, mailing address, look up in the contact manager, search name on the Web, etc. Chaining now adds powerful suggestions. The WhoWhere Web service takes a name as input and returns a Web browser showing a list of possible e-mail addresses corresponding to that name. By making the assumption that the first e-mail address returned in the list is the correct one, we can now use this service to convert the name to an e-mail address. The service now creates related e-mail address data, and the user is supplied with all possible suggestions for a string, a name, and an e-mail address.

### 5.2 Combining Context Data

Even with chaining, the system is still limited to services that can use only one context data type at a time. We can combine the data types to enable the use of services acting on more complex context information. When newly converted data is observed, the combining ability takes the newly converted data and adds it to the original data creating a more complex data object. This new data triggers more powerful actions. Using the previous example of a user reading an appointment in her scheduler, the user selects a name, and a chaining service like Four11 is used to obtain a mailing address for this name. Using combining, a data object containing both the name and the mailing address may now be used as input to a phone number lookup service like Switchboard. Switchboard can find phone numbers when given simply a name as input, but it can perform a more accurate search when it is provided with both a name and a mailing address.

Most services will perform better when provided with pertinent, additional context to work with. CyberDesk determines how to bind data together based on the data it currently has (the sum total of the current user context) and on the services available. It will offer a suggestion to use Switchboard with just a name as input when only name information is available, but will suggest Switchboard with a name and a mailing address if both pieces of information are available.

In a more complete example of combining, the user selects the name of a person she is meeting tomorrow. Immediately, she is offered suggestions of actions that she can perform with the selected string and name. As the chaining applications return their data, this suggested list of actions is augmented with actions that can use an e-mail address (via WhoWhere), phone numbers and mailing addresses (via Switchboard) and URLs (via AltaVista). At the same time, the Integrator is dynamically binding these individual pieces of data for services that benefit from multiple data inputs. The user chooses to create a new entry in the contact manager. This results in a rich entry, containing the original name she selected, an e-mail address, a URL, a phone number, and a mailing address.

## 6 FUTURE WORK FOR MOBILITY

Chaining and combining are two extensions to CyberDesk that arose out of a need to provide better context-aware services. Simple extensions of the dynamic integration mechanism allowed for these improved services. We now want to suggest yet another application of the CyberDesk infrastructure that again shows the merits of the dynamic registration service. The following scenario has not yet been completed in our lab, but it will be soon, and its descriptions helps to show how the software engineering considerations of our framework help to feed different context-aware applications.

### The Intelligent Mobile Display

A possible negative side-effect of CyberDesk is suggestion overload. If enough services are loaded in the CyberDesk system, the user could be overwhelmed with the number of integrating suggestions made at any one time. A further extension to CyberDesk can reduce the number of suggestions made by predicting only the most relevant suggestions. The Registry is dynamic, allowing components to be added and removed at any time. The user's context can aid in the removal of services that are not relevant at the current time and add components whose services should be active given the current context.

We are developing a mobile application that takes advantage of this extension. A sample scenario follows. At noon, a user enters his kitchen and walks up to a mobile computing tablet hanging on his refrigerator. A recognition system determine the identity of the user and his location in front of the refrigerator. The tablet displays a list of suggestions to the user: show a list of items in the refrigerator, search for a recipe that can be made with the items in the refrigerator, or display the latest family pictures. The user picks up the display and moves into the living room and sits on the couch. The positioning system recognizes that the user and tablet have moved into the living room and the system offers the following list of suggestions: view today's news, read personal e-mail messages, or browse an electronic journal. A little later, a business partner drops by and the user moves to his office with his partner. The system recognizes the partner and determines the current location of the display and user. The display now suggests looking up the partner's contact information, viewing business-related newsgroups, viewing notes from the last meeting attended by both the user and the partner.

As can be seen in this scenario, the suggestions offered by the system are tailored to the current location and identification of the people using it. The services suggested to the user are limited by the context of the user and his surrounding environment, thus reducing the potential problem of suggestion overload.

## 7 CONCLUSIONS

In this paper, we have demonstrated the value of software engineering research focussed on applications of future interactive technology. The CyberDesk system is a context-aware framework that provides a novel interface to personal information management, blurring the distinction between desktop, network and mobile services. CyberDesk's extensible context-inferencing layer provides a new way to leverage the knowledge of user context into more automated and personalized services. More importantly, the infrastructure used to create CyberDesk is grounded in software engineering techniques for dynamic, component-based integration. While none of the components of the CyberDesk infrastructure is in itself novel, the combination into a extensible context-aware integration framework is a contribution. Some of the more important features of this infrastructure are:

- Recursive context-inferencing engine based on type converters. This aspect of the framework provides the apparent intelligence of the system. Since type converters are simple Java classes, they can be added or modified to increase the predictive power of the overall system. The simple insight that services can also provide a type conversion interface also opened up the context-inferencing engine to arbitrary conversions that can be found anywhere on the Internet.
- Dynamic mediation between user context and service invocation. The Integrator in CyberDesk allows the run-time observation of user context to trigger relevant information services automatically to the user. This relieves the designer of a particular service of the need to predict integrating behavior at design time. It also allows the user to substitute services within the CyberDesk framework. These are two of the main objectives we had for improving upon existing commercial integration suites.

## ACKNOWLEDGEMENTS

Anind Dey is supported by Motorola Corporation through the University Partnerships in Research (UPR) Program, sponsored by Dr. Ron Borgstahl. The authors would like to thank the members of the Future Computing Environments Group and the numerous other undergraduate and graduate students at Georgia Tech who have provided much inspiration and support in the development of the initial CyberDesk prototype and have offered us a lot of evidence for the scalability of the infrastructure.

## REFERENCES

[1] G. D. Abowd. Ubiquitous computing: Research themes and open issues from an applications perspective. Technical Report GIT-GVU 96-24, GVU Center, Georgia Institute of Technology, October 1996.

[2] G. D. Abowd, C. G. Atkeson, J. Hong, S. Long, R. Kooper, and M. Pinkerton. Cyberguide: A mobile

context-aware tour guide. *ACM Wireless Networks*, 3, 1997. To appear.

[3] G. D. Abowd and B. Schilit. Ubiquitous-computing: The impact on future interaction paradigms and hci research. CHI'97 workshop, March 1997. Materials for the workshop are available via http://www.cc.gatech-.edu/fac/Gregory.Abowd/ubi-workshop/.

[4] Apple Enterprise Software. Topics in openstep programming. Available at http:www.next.com/Pubs-/Documents/OPENSTEP/ProgrammingTopics/, 1997.

[5] Apple Research Laboratories. Apple data detectors homepage. Available at http://www.research.apple-.com/research/tech/AppleDataDetectors/, 1997.

[6] A. Asthana, M. Cravatts, and P. Krzyzanouski. An indoor wireless system for personalized shopping assistance. In L.-F. Cabrera and M. Sattyanarayanan, editors, *Workshop on Mobile Computing Systems and Applications*, pages 69–74. IEEE Computer Society Press, December 1994.

[7] D. G. Bobrow, L. G. DeMichiel, R. P. Gabriel, S. E. Keene, G. Kiczales, and D. A. Moon. Common lisp object system specification x3j13. In SIGPLAN Notices 23 (special issue), September 1988.

[8] A. Dey, G. D. Abowd, and A. Wood. Cyberdesk: A framework for providing self-integrating context-aware services. In *Proceedings of the 1998 Intelligent User Interfaces Conference — IUI'98*, 1998. To appear.

[9] I. Essa and A. Pentland. Facial expression recognition using a dynamic model and motion energy. In *Proceedings of the International Conference on Computer Vision*, pages 360–367. IEEE Computer Society, Cambridge, MA, 1995.

[10] T. Finin, R. Fritzson, and D. McKay. A language and protocol to support intelligent agent interoperability. In *Proceedings of the CE & CALS Washington'92 Conference*, 1992.

[11] G. E. Krasner and S. T. Pope. A cookbook for using the model-view-controler user interface paradigm in smalltalk-80. *Journal of Object-Oriented Programming*, 1(3):26–49, August 1988.

[12] A. C. Long, Jr., S. Narayanaswamy, A. Burstein, R. Han, K. Lutz, B. Richards, S. Sheng, R. W. Brodersen, and J. Rabaey. A prototype user interface for a mobile multimedia terminal. In *Proceedings of the 1995 conference on Human Factors in Computing Systems — CHI'95*, 1995. Interactive experience demonstration.

[13] Microsoft Corporation. Ole development homepage. Available at http://www.microsoft.com/oledev, 1997.

[14] OMG. *The Common Object Request Broker: Architecture and Specification V2.0.* Object Management Group, Inc., formal/97-02-25 edition, July 1996.

[15] M. Pandit and S. Kalbag. The selection recognition agent: Instant access to relevant information and operations. In *Proceedings of Intelligent User Interfaces '97*. ACM Press, 1997.

[16] R. Picard. Affective computing. Technical Report 321, MIT Media Lab, Perceptual Computing, November 1995. Available as MIT Media Lab Perceptual Computing Techreport # 362 from http://vismod.www.media.mit.edu/vismod/.

[17] M. Pinkerton. Ubiquitous computing: Extending access to mobile data. Technical Report GIT-GVU-97-09, GVU Center, Georgia Institute of Technology, June 1997. Master's thesis.

[18] S. P. Reiss. Integration mechanisms in the FIELD environment. Technical Report CS-88-18, Brown University, October 1988.

[19] S. P. Reiss. Connecting tools using message passing in the FIELD environment. *IEEE Software*, 7(4):57–66, July 1990.

[20] W. N. Schilit. *System architecture for context-aware mobile computing.* PhD thesis, Columbia University, 1995.

[21] M. Schneider-Hufschmidt, T. Kühme, and U. Malinowski, editors. *Adaptive User Interfaces: Principles and Practice.* North-Holland, 1993.

[22] K. Sullivan and D. Notkin. Reconciling environment integration and component independence. In *Proceedings of SIGSOFT 90: Fourth Symposium on Software Development Environments.* ACM Press, 1990.

[23] K. J. Sullivan. *Mediators: Easing the design and evolution of integrated systems.* PhD thesis, University of Washington, 1994.

[24] R. Want, A. Hopper, V. Falcao, and J. Gibbons. The active badge location system. *ACM Transactions on Information Systems*, 10(1):91–102, January 1992.

[25] M. Weiser. The computer of the 21st century. *Scientific American*, 265(3):66–75, September 1991.

[26] M. Weiser. Some computer science issues in ubiquitous computing. *Communications of the ACM*, 36(7):75–84, July 1993.

[27] G. Wiederhold. Mediators in the architecture of future information systems. *IEEE Computer*, pages 38–49, March 1992.

[28] A. Wood, A. Dey, and G. D. Abowd. Cyberdesk: Automated integration of desktop and network services. In *Proceedings of the 1997 conference on Human Factors in Computing Systems — CHI'97*, pages 552–553, 1997. Technical note.

[29] A. M. Wood. CAMEO: Supporting observable APIs. Position paper for the WWW'5 Programming the Web Workshop, May 1996.