CyberDesk: A Framework for Providing Self-Integrating Context-Aware Services

Anind K. Dey, Gregory D. Abowd
Graphics, Visualization, & Usability Center,
Georgia Institute of Technology
Atlanta, GA 30332-0280 USA
+1-404-894-7512
email: {anind, abowd }@cc.gatech.edu

Andrew Wood
School of Computer Science
The University of Birmingham
Edgbaston, Birmingham, B15 2TT UK
email: amw@cs.bham.acuk

ABSTRACT

Applications are often designed to take advantage of the potential for integration with each other via shared information. Current approaches for integration are limited, effecting both the programmer and end-user. In this paper, we present CyberDesk, a framework for self-integrating software in which integration is driven by user context. It relieves the burden on programmers by removing the necessity to predict how software should be integrated. It also relieves the burden from users by removing the need to understand how different software components work together.

Keywords

Context-aware computing, automated software integration, dynamic mediation, ubiquitous computing

INTRODUCTION

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, and Netscape Communicator).

We recognize the utility from the user's perspective of integrating the behavior of a number of software applications. With the emergence of Web-based applications and personal digital assistants (PDAs), there are even more opportunities to provide integration of software applications. There are some limitations, however, to the current approaches for providing this integration. These limitations impact both the programmer and the user.

IUI 98 San Francisco CA USA
Copyright 1998 ACM 0-89791-955-6/98/ 01..$3.50

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. This means that a programmer has the impossible task of predicting all of the possible ways users will want a given application to work with all other applications. This usually results in 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. 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.

The project described in this paper, CyberDesk, is aimed at providing a more flexible framework for integrating software behavior. 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. The main objective of the ubiquitous computing project CyberDesk is to provide the infrastructure for self-integrating software in which the integration is driven by actions of the user. We refer to this as context-aware integration, and it is aimed at producing a paradigm shift in human-computer interaction that is fundamental to ubiquitous computing. Rather than settle for the current situation, in which the user must seek out and find relevant software functionality when she wants it, we instead want the ubiquitous computing infrastructure to seek out the user when and where she wants it. In this paper, we demonstrate how CyberDesk supports this paradigm shift.

WHAT IS CYBERDESK?

CyberDesk is a component-based framework written in Java, that supports automatic integration of software applications. The framework is flexible, and can be easily customized and extended. The components in CyberDesk treat all data uniformly, as a Java object, regardless of whether the data came from a desktop application, a PDA-based application, or a Web-based application.

The intelligence in CyberDesk's user interface comes from applications automatically providing their services to the user. Rather than displaying all the services to the user at all times, the interface is limited to those services that are relevant to the user's context. A service is an action that an application can perform or data that an application can provide. A user's context is any information about the user and the environment that can be used to enhance the user's experiences. This includes the data the user is working with, the time of day, the user's physical location, emotional state, social environment, objects in the room, etc. Initially, CyberDesk was only able to work with simple strings that a user was working with in a desktop application. Now, CyberDesk is also able to work with time and location in a desktop environment, networked environment, and mobile environment.

Desktop applications incorporated into CyberDesk include e-mail browsers, notepads, schedulers, and contact managers. Network applications include phone number lookups, e-mail writing, mailing address lookups, Web searches, Usenet searches, e-mail address lookups, map lookups, and Web page browsing. PDA-based applications include contact managers and notepads. All applications make their services available to the user via a common interface. The services available at any particular time depend on the user's context at that time. By providing relevant suggestions and data to the user, the user receives useful, and possibly unexpected, help in completing their tasks.

User Scenario

To illustrate this behavior, an actual user experience follows.[1] As seen in Figure 1, a user is checking his e-mail, and reads one about some interesting research.

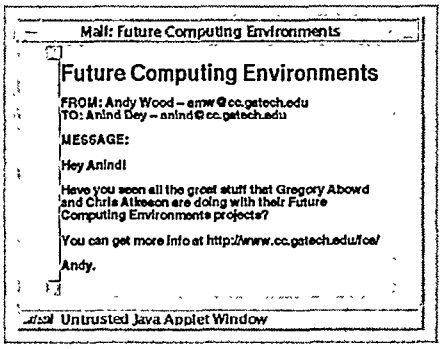

Figure 1. Content of User's E-mail Message.

The user is interested in the research discussed, highlights the URL in the message, and CyberDesk offers the following suggestions through its interface (Figure 2): search for the selected text using AltaVista, find pages that reference this URL using AltaVista, and display the URL in Netscape.

[1] All the scenarios described in the paper can be executed at http://www.cc.gatech.edu/fce/cyberdesk/iui.

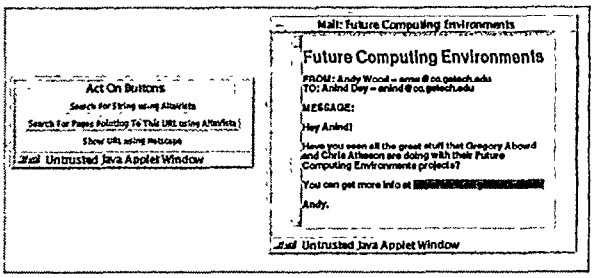

Figure 2. User selects the URL and is offered suggestions.

He chooses the last option and views the URL listed in the message (Figure 3).

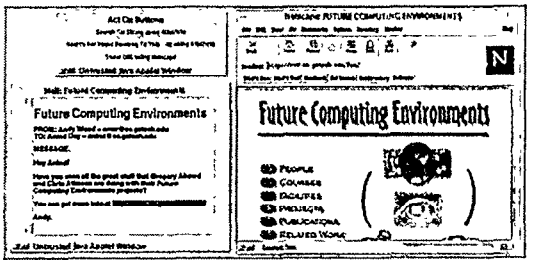

Figure 3. CyberDesk executes the service and displays the URL.

The user then selects the name of the person in charge of the research and is offered the following suggestions (Figure 4): search for the selected text using AltaVista, search for a phone number and mailing address using Switchboard, lookup the name in the contact manager. The user wants to contact this researcher so he checks to see if the name is in his contact manager, but it isn't.

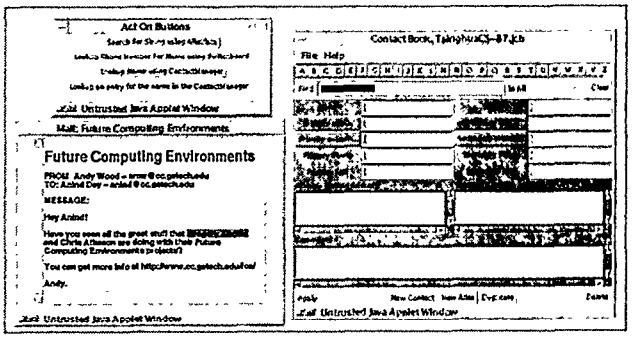

Figure 4: User selects name and chooses the Contact lookup service.

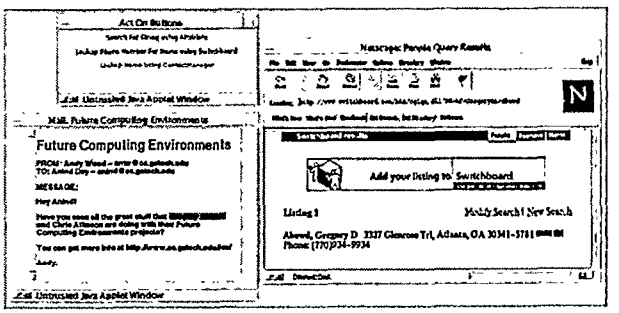

Figure 5. User selects the phone number lookup service.

So, he selects the phone number and mailing address lookup service (Figure 5). He then creates a new entry in the contact manager with this new information.

ARCHITECTURE

The CyberDesk system has a simple architecture, based on an event-driven model, where components act as data sources and/or data sinks. The system consists of four core components: the IntelliButton, the ActOn Button Bar, the applications, and the type converters. The IntelliButton maintains the registry of data sources and sinks. It also finds services or data sinks that match the input data, a task normally required of the system or service designer. The IntelliButton displays the matches in the form of suggestions to the user, via the ActOn Button Bar. It is through the ActOn Button Bar that the user accesses the integrating functionality of CyberDesk. The applications are the data sources and sinks themselves, and are the tools the user ultimately wants to use. When the user's context changes, either by working with new data or by a change in time or position, the new context is passed to the CyberDesk system (Figure 6). The type converters provide more powerful integrating behavior by converting this data (a string in the previous scenario) into other data types (e.g. name using regular expression matching), allowing for a greater number of matches.

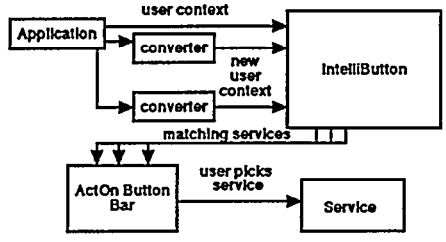

Figure 6. Runtime architecture diagram.

All of the components have been implemented as Java applets for simplicity of network programming. We also chose Java for its promise of platform independence, ability to execute within a Web browser, and object-oriented nature. The first two features support our goal of ubiquity, and the last feature made development easier. Also, most of the integrated network applications are available via the Web, so the natural access method was via a Web browser. Inter-component communication was performed using techniques based on the CAMEO toolkit [22], a C++ toolkit built previously by one of the authors to facilitate the integration of application-sized components via the use of agent-like components. More information on CyberDesk's architecture can be found in [7].

ADDING APPLICATIONS TO CYBERDESK

As discussed earlier, the applications used in CyberDesk are the actual tools the user wants to use. CyberDesk provides an easier and faster way of accessing the functionality of these applications and the data they contain (i.e. their services). From the user's perspective, adding an application (or any CyberDesk component) to CyberDesk simply requires the addition of HTML applet tags to a CyberDesk HTML page. From the programmer's perspective, adding an application requires more effort.

Currently, CyberDesk is unable to automatically determine the services each application provides. A service programmer must construct a wrapper around each application. This wrapper performs two main functions: registration of the provided services with CyberDesk and execution of the services when called. During the registration process, each application registers with the IntelliButton giving a list of services it provides, both actions it can perform on different data types and the data types it can produce. Examples of this from the previous scenario are:

- the AltaVista wrapper declaring it can search the Web for a string and find pages that reference a given URL
- the Contact Manager wrapper declaring it can lookup a given name, create a new entry, and can produce string objects when a user selects data in the contact manager.

The second portion of the wrapper deals with actually executing the services that were registered. When the user selects a service from the interface, a method in the wrapper is called to execute the service. This method takes the user's context, retrieves the relevant parameters for the service and calls a method that will execute the user-selected service.

All the Web-based applications employed by CyberDesk use HTML forms. By analyzing the form, a programmer can fairly easily write the wrapper. These applications generally have straightforward interfaces and require a small set of input parameters. The parameters are passed to a URL, which generates a resulting HTML page that can be displayed in a Web browser. For example, the AltaVista Web search service simply requires an input string and returns a list of all Web pages that match this string. Currently, there are 68 Web-based applications that have been integrated into CyberDesk. A service writer program has been written to automatically generate a wrapper for Web-based applications. This program is intended for use by service programmers, but is simple enough to be used by an end-user wanting to add a service to CyberDesk. The program takes a URL containing a form as input and presents an interface as shown in Figure 7.

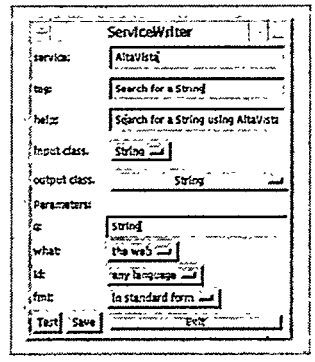

Figure 7. Service Writer interface.

The service programmer selects the data type the service can

act on, the values for the service parameters, and the output data type, if any. Upon receiving this information, the service writer program generates a wrapper for the service.

Other Georgia Tech students have written all of the desktop applications added to CyberDesk. This provided access to both the application APIs and the source code. The APIs were needed to determine the names of methods for services the applications provided and the parameters that each method required. The source code allowed the service programmers to add additional services to the applications and add the data selection ability shown in the original scenario (e.g. user selecting the URL). It should be made clear that source code was only modified to enhance the existing applications. If only the API were available for an application, and not the source code, a wrapper could still be written to take advantage of the application's existing functionality. Currently, there are 6 desktop applications that have integrated into CyberDesk.

In addition, an automated service writer, similar to the one mentioned above, is being designed. The service writer will be based on the newest release of the Java language (version 1.1). It provides an automatic data selection feature, allowing for the transfer of data between (Java and non-Java) applications via a clipboard-style interface, and supports the use of reusable software components called JavaBeans [9]. The data transfer feature will eliminate the need for any application source code. The use of JavaBeans would allow the service writer to query an application and determine its API at run-time, removing the need for a compile-time API and allowing the use of third party applications.

ADDING TYPE CONVERTERS TO CYBERDESK

CyberDesk applications can generate changes in the data (generally strings) the user is working with. As described above, CyberDesk uses type converters to convert this user context (or location or time information, as will be shown in an upcoming section) into other useful forms of user context. For example, in the user scenario, a StringToURL converter took the data selected by the user and successfully converted it to a URL. This resulted in two pieces of data being sent to the IntelliButton, a string and a URL. The IntelliButton sought integrating behavior for both these types, allowing the user to access URL-relevant services where originally they wouldn't have had the option.

The type converters work in a recursive fashion. That is, the new data that is generated from a successful conversion is sent to the type converters. This process continues until no new data is created, or a cycle is found.

Initially, applications were hardcoded to generate different data types. For example, the e-mail browser declared that it could generate strings when text is highlighted, but also EmailAddress objects when the "To:" or "From:" field in an e-mail message was selected. When EmailAddress objects were generated, they were passed through the CyberDesk system, as described before, to the ActOn Button Bar, which displayed services that could consume EmailAddress objects (e.g. Send an E-mail to this E-mail Address using Netscape). However, this required the applications themselves to be aware of the CyberDesk type system. It was also limiting since e-mail addresses could also appear in the unformatted body text of an e-mail message and only be recognized as a string selection.

Consequently, a decision was made to use type converters. Using simple heuristics, it is possible to identify potential text strings that might be e-mail addresses. It would have been desirable to augment the e-mail browser with this capability, so that any time text was selected in it, it would try to convert the text to an e-mail address and create an EmailAddress object rather than just a string. But, instead of just giving this type conversion capability to the e-mail browser, that ability should be added to the system once, and allowed to be used in every application where e-mail addresses might appear. The type detection ability was removed from the individual applications and type converters, an independent and extensible layer in the architecture, were created.

For the programmer, writing a type converter involves writing a method that accepts one data type and converts it to another. For the user, adding type converters to a CyberDesk session allows for the use of a wider variety of user context. When user context changes, (change in time, location, or data selection) type converters improve the list of suggested actions given by CyberDesk by providing services specific to the content of the user context, not relying simply on the type of user context that has changed.

Currently the list of CyberDesk types includes Date, PhoneNumber, MailingAddress, Name, URL, EmailAddress, GPSPosition, and Time. For each data type, there is a corresponding StringTo (data type) type converter.

CHAINING

By making a simple extension to the application wrappers, applications can gain the same advantages as type converters. Most of the services provided by applications require data types as input parameters and display their results through a graphical interface. This was seen in the above scenario when the user searched for a phone number and mailing address (a Web page was displayed) and when the user looked for a name in his contact manager (a contact entry was displayed). Through the use of simple parsing, this data encoded in the graphical interface can be obtained. In the phone number search, the HTML page returned can be examined and parsed to retrieve a matching phone number. Similarly, if the name being looked up in the contact manager had an existing entry containing an e-mail address, the e-mail address could be easily retrieved. This new data is part of the user's context and is made accessible to CyberDesk.

When applications are able to generate additional pieces of context and perform user-selected actions, they are behaving both as type converters and as applications, providing the

advantages of both. Now, applications can asynchronously suggest both actions directly related and indirectly related to the change in user context, reducing the effort required by the user to find these services. This process of generating additional context for the purpose of increasing integrating behavior is called chaining.

A sample user scenario is described below. A user is reading an appointment in her scheduler and selects the name of the person she is supposed to be meeting (Figure 8). 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. However, by using chaining, more powerful suggestions can be had. The WhoWhere Web application takes a name as input and returns a Web browser showing a list of possible e-mail addresses corresponding to that name. If we make the assumption (not always a good one) 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 a related EmailAddress object, and the user is supplied with all possible suggestions for both a Name and an EmailAddress.

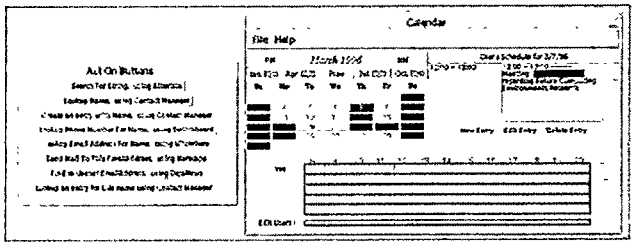

Figure 8. Chaining example.

Chaining is potentially a very powerful tool for the user to take advantage of. It provides another dimension of suggestions for each data type that the user context can be converted to.

COMBINING

Along the same line of thought, chaining can be used along with the concept of combining to make services more powerful. The services previously described were designed to only operate on a single data type (at a time). With data being converted to multiple types via chaining, the idea is that the services should be able to take advantage of these multiple types. They can, through a process we call combining.

Combining, in CyberDesk terms, is the ability to collect multiple data types and dynamically bind them together, as needed, to create meta-objects which services can use. These meta-objects can be used to perform substantially more powerful actions. Using the above 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 (and create a related MailingAddress object) for that name. Using combining, a meta-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, but it can perform a more accurate search when 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 not offer a suggestion to use Switchboard with just a name as input, when it can suggest it with both a name and mailing address.

Now that the concept of combining has been explained, a more complete example demonstrating its power is given below. Again, we'll use the example of the user reading an appointment in her scheduler (Figure 9). She 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-enhanced applications return their data, this suggested list of actions is asynchronously 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 IntelliButton is dynamically binding these individual pieces of data for services that benefit from multiple data inputs.

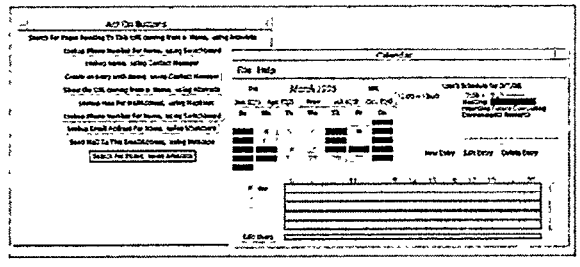

Figure 9. Combining example - user selects a name and is offered many integrating suggestions.

The user chooses to create a new entry in the contact manager. This results in a rich entry (Figure 10), containing the original name she selected, an e-mail address, a URL, a phone number, and a mailing address.

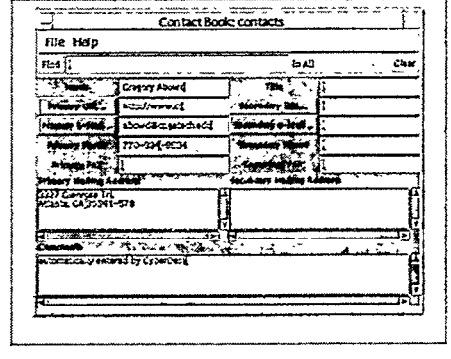

Figure 10. Combining example - user creates rich contact entry.

Combining, like chaining, can be very powerful to the user. It does not inhibit the list of options for individual pieces of

user context, while at the same time it combines those pieces, enhancing the available services and their results. In essence, chaining and combining create a context inference engine in CyberDesk.

OTHER FORMS OF CONTEXT

In a mobile setting, there are additional forms of context that are not necessarily available in a desktop environment. Examples include a user's changing location, the changing objects in the environment, and the familiarity with the environment. CyberDesk has integrated services that allow access to data and applications on mobile devices [16] (a Newton MessagePad, in particular). Now, we are looking at integrating services that are available when the user is mobile, to take advantage of these other forms of context.

Up until now, all of the examples shown have only used changes in context based on the data the user is currently working with. CyberDesk can deal with other forms of context in the same way as it deals with the user's data. Integrated examples include significant changes in time and position. One application that has been added to CyberDesk is one that updates the system time every five minutes. Currently, only one service that can use time has been integrated into CyberDesk. This service is part of the scheduler and it acts as a reminder service for events listed in the scheduler. When the time input into the scheduler is within fifteen minutes of an event, the scheduler offers a suggestion to the user to check their scheduler. Another, but more intrusive, option would be to create a window displaying the relevant information to the user. Ideally, the user would be able to set the type of feedback desired, and the event windows (i.e. how often the time service updates and how close to an event should a user be warned).

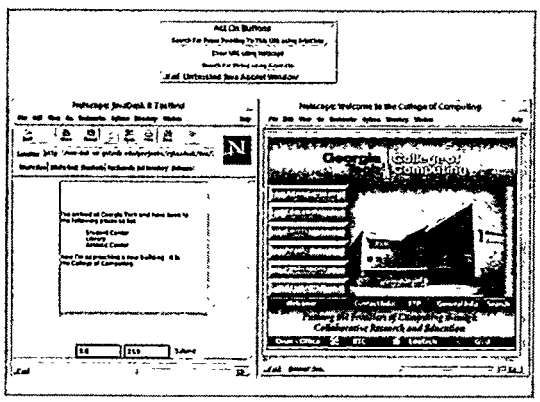

Figure 11. Screenshot of position service. (a) is where GPS coordinates are being input, causing changes in the ActOn Button Bar (b) when the coordinates correspond to a different Georgia Tech building. The user is keeping track of his trip in the scratchpad (c), and is able to view the building URLs in the Web browser (d).

Position information has also been incorporated into CyberDesk, for use in a mobile setting. The current system uses Global Positioning System (GPS) data and is intended for outdoor use. The application providing GPS data updates the system position whenever the GPS coordinates change. Again, how often the application updates will be a user-controlled parameter. A service has been written that accepts GPS information for a location on the Georgia Tech campus and returns a URL corresponding to that location. CyberDesk then suggests all the activities it can perform with a URL, including displaying it in a Web browser. An example of this is shown in Figure 11.

A prototype of an indoor positioning system has been built at Georgia Tech [11]. This prototype could be used as an application offering information on a user's location within a building, updated as a user moves between rooms. Possible services that incorporate both types of positioning information are real-time mapping and directions, access to equipment in the environment, and providing information on important landmarks (washrooms, ATMs, etc.). If these services were combined with knowledge of a user's history, the services could be made even more useful to the user. When a user approaches a building or room they've never been to, the CyberDesk system should offer introductory information on the location. If the user has been there before, different sets of information should be offered.

Additional forms of context can be used to generate new and more appropriate suggestions to the user. As CyberDesk's knowledge of the user's context grows, it is able to create more informed suggestions for the user. We are interested in using CyberDesk as the basis for context-aware applications that we are developing - applications that take advantage of knowing a user's position, history, behavior, etc. While there has been a lot of research in context-aware applications [1,17,21], we are not aware of a general toolkit which supports such a wide variety of user context and integration behavior like CyberDesk does.

BACKGROUND AND RELATED WORK

The underlying framework of CyberDesk that allows integration of isolated services is based on the concept of dynamic mediation. Mediation consists of two basic steps: registration of components and handling of events. Other systems that use mediation include UNIX pipes, Field [8], Smalltalk-80 MVC [10], Common Lisp Object System (CLOS) [4], and APPL/A [20]. 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. Like CyberDesk, it 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, much like CyberDesk. But it limits the action a component can perform to a simple method call and return, thereby limiting its usefulness. Sullivan and Notkin [19] 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, while CyberDesk allows one-to-many relationships and allows a looser, more flexible, registration process.

CyberDesk also depends on the use of component software and network objects. These concepts are important for system flexibility and reuse. Other systems that provide for these concepts are CORBA (Common Object Request Broker Architecture) and IIOP [13] (Internet Inter-ORB Protocol), IBM's DSOM [12] (Distributed System Object Model) and Microsoft's OLE and DCOM [12]. These are object models that allow cross-network and cross-language integration of applications.

There are three systems that provide functionality similar to CyberDesk. They are OpenStep's services facility [14], Intel's Selection Recognition Agent [15], and Apple Research Lab's Data Detectors [2].

The OpenStep computing environment uses a uniform object-messaging interface between objects in all of its applications, similar to CyberDesk. Using this ability, applications can declare the types of data they can generate and are integrated with services that can operate on that data. The largest difference between CyberDesk and OpenStep services is that CyberDesk acts on both the content and data type being used, rather than just the data type. OpenStep services are primarily used to convert file formats, create dynamic links between objects (i.e. updating an object updates the linking documents), and providing global services such as spell checking and printing. It works only with data the user is attending to, limiting the context types it can use, and does not support the concepts of chaining or combining.

Intel's Selection Recognition Agent attempts to address the same issues as CyberDesk. Unlike CyberDesk, it uses a fixed data type-action pair, allowing for only one possible 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. It does not support any of the advanced features of CyberDesk, like chaining or combining, nor does it use any other forms of context like time or position.

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. Like CyberDesk, it supports an arbitrary number of actions for each data type. It does not support chaining and supports only a very limited notion of combining. 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 Apple Data Detectors system does not support the use of other forms of context. Its focus appears to be desktop applications, as opposed to CyberDesk's ubiquitous services, existing either locally or remotely.

ISSUES AND FUTURE WORK

The CyberDesk framework was designed to be easily extensible and easy to use, however it suffers from a few limitations. There is still a programming burden involved with integrating services into CyberDesk. This process is not yet entirely automated. This issue is being addressed through the efforts of the Web-based service writer and the investigation of JavaBeans.

CyberDesk is still limited by the number of different types of user context it utilizes. Note that this is not a limit imposed by the CyberDesk infrastructure. The use of history, personal preferences, and the location of physical objects and landmarks is currently being examined for integration into CyberDesk.

Perhaps the biggest limitation of the system is the user interface implemented by the ActOn Button Bar. It consists of a window that displays a list of suggested user actions. Although the system looks for repeated suggestions, it is clear that the number of possible suggestions could quickly become overwhelming to the user. Possible methods for limiting the number of suggestions are:

- before displaying a suggestion, contact the service corresponding to the suggestion and ensure that it can successfully perform the action
- pass the service name along with the data it generates, to see if the service has already acted on data
- use user history and preferences to select suggestions most likely to be accepted
- use context to filter out suggestions

For example, if a user selects a name in an e-mail message and the system knows that the name is not in the contact manager, CyberDesk should not offer a suggestion to lookup the name in the contact manager, but instead should suggest to create a new entry in the contact manager. Another example is when the system has access to a user's history, it could determine the most likely actions a user is likely to take, and limit the suggestions to those, or at least order the suggestions accordingly.

We are currently looking at different ways to adapt the interface to initially show actions that the user is likely to take, while providing a way for the user to see other possible actions as well. We are also looking at different presentation methods for the suggestions, including pop-up hierarchical

menus, having menus associated with each individual application, and document lenses [3].

CyberDesk has also shown the potential for supporting more complex forms of context. For example, if an e-mail message contains information about a meeting, and the user selects the message content, a type converter could potentially convert the text to a Meeting object to be inserted in the user's Calendar Manager. Of course, retrieving context from arbitrary text is a very difficult problem being investigated by the AI learning community. But the power of CyberDesk supports the ability to use this higher level context, if available.

We will continue to add services to expand CyberDesk's library of components but this will not be our main focus. We are more interested in the following research areas:

- examining the use of advanced techniques like chaining and combining and searching for others.
- investigating learning-by-example techniques [6] to allow the CyberDesk system to dynamically create chained suggestions based on a user's repeated actions.
- incorporating rich forms of context into CyberDesk, other than time, position, and meta-types. This will allow us to use CyberDesk as the platform for developing context-aware, mobile applications.
- experimenting with adaptive interfaces and different interface representations in order to determine better ways of presenting suggestions to our users.

CONCLUSIONS

Providing intelligence in a ubiquitous environment can be achieved by taking advantage of the user's context. Context includes the information a user interacts with on a desktop or mobile device, location, time, etc. We have developed a framework for integrating software services based on a user's context. CyberDesk eases the burden on programmers by relieving the necessity to determine all integrating possibilities and eases the burden on users by relieving the necessity to understand how applications work together. Through the use of advanced techniques like chaining and combining, we have shown the potential for integrating behavior that is too complicated for a programmer to statically design. Context-aware integration changes the paradigm of interaction from a user seeking out functionality in software applications to the infrastructure seeking out the user at relevant times.

REFERENCES

1. Abowd, G. et al. Context-awareness in wearable and ubiquitous computing. Technical Report, GVU Center, Georgia Institute of Technology. GIT-GVU-97-22, 1997.
2. Apple Research Labs. Apple Data Detectors homepage. Available at http://www.research.apple.com/research/tech-/AppleDataDetectors/.
3. Bier, E.A. et al. ToolGlass and Magic Lenses: The See-Through Interface. Computer Graphics Proceedings, Annual Conference Series, 1993. ACM SIGGRAPH. 73-80.
4. Bobrow, D. et al. Common Lisp Object System Specification X3JI3 Document 88-02R. ACM SIGPLAN Notices 23. September 1988.
5. Brando, T. Interoperability and the CORBA specification. MITRE Document MP-95B-58. February, 1995.
6. Cypher, A. EAGER: Programming repetitive tasks by example. In Proceedings of CHI' 91. ACM Press.
7. Dey, A.K., et al. CyberDesk: A Framework for Providing Self-Integrating Ubiquitous Software Services. Technical Report, GVU Center, Georgia Institute of Technology. GIT-GVU-97-20, 1997.
8. Garlan, D. et al. Low-cost, Adaptable Tool Integration Policies for Integrated Environments. Proceedings of SIGSOFT 90: Fourth Symposium on Software Development Environments. Irvine, CA, 1990.
9. JavaSoft. JavaBeans homepage. Available at http://splash.javasoft.com/beans/.
10. Krasner, G. et al. A Cookbook for Using the Model-View-Controller User Interface Paradigm in Smalltalk-80. Journal of Object Oriented Programming 1,3 (August/September 1988), pp. 26-49.
11. Long, S. et al. CyberGuide: Prototyping Context-Aware Mobile Applications. In Proceedings of CHI '96 (Vancouver, Canada, March 1996), ACM Press.
12. Microsoft. OLE Development homepage. Available at http://www.microsoft.com/oledev.
13. Object Management Group homepage. Available at http://www.omg.org.
14. OpenStep. Topics in OpenStep Programming. Available at http://www.next.com/Pubs/Documents/OPENSTEP/-ProgrammingTopics.
15. Pandit, M. and Kalbag, S. The Selection Recognition Agent: Instant Access to Relevant Information and Operations. In Proceedings of Intelligent User Interfaces '97. ACM Press.
16. Pinkerton, M. Ubiquitous Computing: Extending access to mobile data. Masters Thesis, Georgia Institute of Technology, June, 1997.
17. Schilit, B. A Context-Aware System Architecture for Mobile Distributed Computing. Ph.D. Thesis, Columbia University. 1995.
18. SOM Object homepage. Available at http://www.software.-ibm.com/ad/somobjects/.
19. Sullivan, K. et al. Reconciling Environment Integration and Component Independence. Proceedings of SIGSOFT 90: Fourth Symposium on Software Development Environments. Irvine, CA, 1990.
20. Sutton, S. et al. APPL/A: A Prototype Language for Software Process Programming. University of Colorado Technical Report CU-CS-448-89, University of Colorado, Boulder, 1989.
21. Want, R. et al. An Overview of the PARCTAB Ubiquitous Computing Experiment. IEEE Personal Communications 2 (6). 1995. 28-43.
22. Wood, A. CAMEO: Supporting Observable APIs. Position Paper for the WWW5 Programming the Web Workshop. (Paris, France, May, 1996).