CyberDesk: A Framework for Providing Self-Integrating Ubiquitous Software Services

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

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

ABSTRACT
Current software suites suffer from problems due to poor integration of their individual tools. They require the designer to think of all possible integrating behaviours and leave little flexibility to the user. In this paper, we discuss CyberDesk, a component software framework that automatically integrates desktop and network services, requiring no integrating decisions to be made by the tool designers and giving total control to the user. We describe CyberDesk's architecture in detail and show how CyberDesk components can be built. We give examples of extensions to CyberDesk such as chaining, combining, and using higher level context to obtain powerful integrating behaviours.

Keywords
Adaptive interfaces, automated integration, dynamic integration, software components, context-aware computing, future computing environments, ubiquitous services

INTRODUCTION
Users are tired of using monolithic application suites that allow little to no customization, just because they are industry standards. Tightly integrated suites of tools/services currently available are unsatisfactory for three reasons. First, they require designers to predict how users will want to integrate various tools. Second, they force users to either be satisfied with design decisions or program their own additional complex relationships between the tools. Finally, users must be satisfied with the available services themselves, because they are often given no opportunity to replace or add services.

In response, software companies have been adopting the notion of component software: using small software modules as building blocks for a larger application. While there are many competing standards (OLE [11], Active X [10], Java Beans [6], OpenDoc [1]), the prevailing view is to provide a framework which programmers and sophisticated users can build upon to create desired application suites.

Unfortunately, current component solutions do not entirely relieve the burden from the designer and end user. Designers must still predict how users will want to integrate various services, without knowing what services the user will have. Designers must also build services specifically for a particular component solution, rather than build a general solution that can be used in multiple frameworks. Users now have the ability to replace and add services at will, but are still forced to accept the integration behaviour of services implemented by the designer.

In this paper, we present the CyberDesk system, a component software framework that relieves most of the burden of integrating services from both the designer of individual services and the end user, provides greater flexibility to the user, and automatically suggests how independent services can be integrated in interesting ways. We begin by giving a short description of CyberDesk and presenting a sample scenario showing how the system could be used. Next, we discuss the architecture underlying the framework and describe the benefits of our system. We end by showing how CyberDesk is being extended to provide more powerful integration behaviour and by describing our future plans.

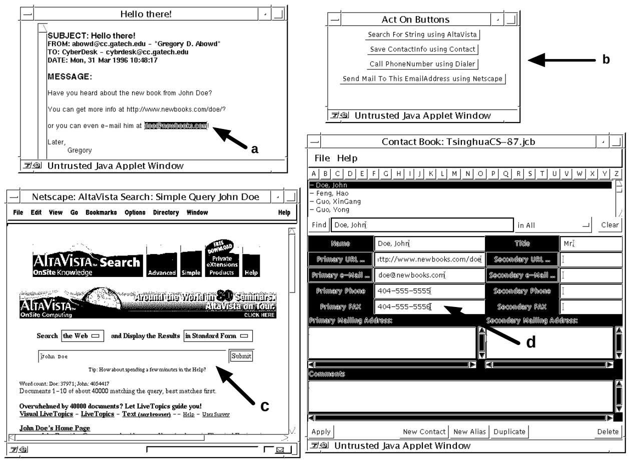
WHAT IS CYBERDESK?
CyberDesk is a component-based framework written in Java, that supports automatic integration of desktop and network services [16]. The framework is flexible, and can be easily customized and extended. The components in CyberDesk treat all data uniformly, regardless of whether the data came from a locally running application or from a service running on the World Wide Web (WWW). The services and applications themselves can be running anywhere, meeting CyberDesk's goal of providing ubiquitous access to services.

User Scenario
The user selects which applications/components they would like to use by adding them to a Hypertext Markup Language (HTML) page. He loads the HTML page into a web browser running on his mobile computer and starts to interact with the system.[1]

The user walks to a grocery store, and the system asks if he wants to see his shopping list, get more information about the grocery store, or get directions to his house. The user chooses the grocery list and goes shopping. He walks to a friend's house but nobody is home. The system asks if he

[1] A demo version of CyberDesk is available at http://www.cc.gatech.edu/fce/cyberdesk. The video accompanying the paper summarizes CyberDesk and shows more sample scenarios. Code samples are available at http://www.cc.gatech.edu/fce/cyberdesk/samples.

wants to check his friend's calendar, contact him via e-mail or phone, or get directions to go home. The user chooses the first option and the system tells him that his friend is at work. So, he chooses the second option, sends his friend an e-mail saying that he stopped by, and starts walking home. On the way home, the system notifies him that he has received an e-mail from his friend. The user reads the e-mail (see Figure 1 below) which has information on a new book written by his favourite author. The e-mail contains a Web site address and an e-mail address for the author. The user highlights the e-mail address (a) and the system gives him some suggestions (b) on what he can do: search for more information on the author, put the author's contact information in the contact manager, call the author, or send an e-mail to the author. He chooses the first two options (c and d), saves the e-mail, and heads home.


Hello there!

SUBJECT: Hello there!
FROM: abowd@cc.gatech.edu - "Gregory D. Abowd"
TO: CyberDesk - cybrdesk@cc.gatech.edu
DATE: Mon, 31 Mar 1996 10:48:17

MESSAGE:

Have you heard about the new book from John Doe?

You can get more info at http://www.newbooks.com/doe/?

or you can even e-mail him at doe@newbooks.com!

Later,
Gregory

Untrusted Java Applet Window

Act On Buttons

Untrusted Java Applet Window

Contact Book: TsinghuaCS-87.jcb

- Doe, John
- Feng, Hao
- Guo, XinGang
- Guo, Yong

Find: Doe, John — in All

Name	Doe, John	Title	Mr
Primary URL ...	ttp://www.newbooks.com/doe	Secondary URL ...	
Primary e-Mail ...	doe@newbooks.com	Secondary e-Mail ...	
Primary Phone	404-555-5555	Secondary Phone	
Primary FAX	404-555-5556	Secondary FAX	

Primary Mailing Address:
Secondary Mailing Address:
Comments

Untrusted Java Applet Window

Netscape: AltaVista Search: Simple Query John Doe

Search the Web and Display the Results in Standard Form

John Doe

Tip: How about spending a few minutes in the Help?

Word count: Doe: 37971; John: 4054417
Documents 1-10 of about 40000 matching the query, best matches first.

Overwhelmed by 40000 documents? Let LiveTopics guide you!
Visual LiveTopics - LiveTopics - Text (any browser) -- Help - User Survey

John Doe's Home Page

a
b
c
d
Search For String using AltaVista
Save ContactInfo using Contact
Call PhoneNumber using Dialer
Send Mail To This EmailAddress using Netscape
File Help
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
Clear
Apply
New Contact
New Alias
Duplicate
Delete
File Edit View Go Bookmarks Options Directory Window Help
AltaVista Search OnSite Knowledge
Advanced
Simple
Free Download Private eXtensions Products
Help
AltaVista OnSite Computing
Around the World in 80 Seminars. AltaVista on Tour. CLICK HERE
Submit

Figure 1. Mock screenshot of above user scenario

The scenario described has not been completely realized with the CyberDesk system. Although, every action and suggested action in the scenario can be realized and supported using the CyberDesk framework. We will show how CyberDesk can support these complex interactions without requiring effort by the user or the system designer.

ARCHITECTURE

The CyberDesk system has a simple but innovative architecture. It is based on an event-driven model, where components act as event sources and/or event sinks. Events, in this current version, are generated from explicit user interaction with the system. The system consists of five core components: the Locator, the IntelliButton, the ActOn Button Bar, the desktop and network services, and the type converters. The Locator maintains the registry of event sources and sinks. This allows the IntelliButton to automatically find matches between event sources and event sinks based on a given input event, 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 services are the event sources and sinks themselves, and are the tools the user ultimately wants to use. The type converters provide more powerful integrating behaviour by converting given events into other events, allowing for a greater number of matches. The five components are discussed in greater detail below.

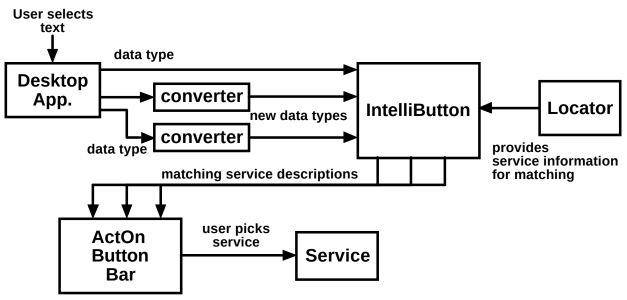


Figure 2: Runtime architecture diagram

All five of the components have been implemented as Java applets for simplicity of network programming. We also chose Java for its promise of platform independence, its ability to execute within a web browser, and its object-oriented nature. The first two features support our goal of ubiquity, the second feature allows us to treat the browser as our desktop [3], and the last feature made development easier. Also, most of the network services implemented 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 [15], 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. Components are able to invoke methods of other components directly via the use of a component handle. The parameter passed in these method calls is a structured message of the following form:

:sender	<*id*>
:receiver	<*id*>
:interface	<*array of interface names*>
:property	<*array of event/status names*>
:arguments	<*data*>

The first two fields contain the object handles for the method caller and method callee, respectively. The interface field refers to the types of actions the component supports. Components declare their ability to be event sinks and sources via the interface field. This will be discussed further in the following section. The types of events that a component consumes or generates is stored in the property field. Data associated with events is passed in the arguments field.

Locator

The Locator component in CyberDesk keeps a directory of all the other components in the system, what events they can generate and/or what events they can consume. In any system where an arbitrary number of components (where types and location are unknown at compile time) are going to be interacting, a method of communication is required; in other words, a rendezvous mechanism that provides introductions between components is needed.

Typically, such a directory service is run at a well-known location. In CyberDesk, the Locator is implemented as a uniquely named applet on an HTML page containing all the CyberDesk applets in use. Upon startup, each of the component applets register themselves with the Locator. It behaves as a yellow pages directory by allowing any component to request a list of all the components supporting a particular interface and property. We currently support two different interfaces: method and select. If the interface field is set to "method", the component contains a method(s) that will consume a particular event type. If the interface field is set to "select", the component is declaring that it can generate a particular event type. Note that a component can support multiple interfaces, in any combination of selects and methods.

The Locator supports the following API:

insert (component_name, interfaces[])

adds a component's interfaces to the registry

remove (component_name, interfaces[])

removes a component's interfaces from the registry

locate (component_name, interfaces[])

locates and returns all components matching a given interface(s)

IntelliButton

The IntelliButton component is really the core of the CyberDesk system, as it provides the automatic integrating behaviour. It uses the Locator to keep track of all the desktop and network services and the type converters, and all the events sources and sinks they provide. When new components are added to the system, the IntelliButton notifies them that it is interested in all the events that they can generate (i.e. it is an event sink). So when a component generates an event, it notifies the IntelliButton and any other components that have expressed interest. The interested components are called observers, as they observe events in other components. Any component can observe multiple components and can be observed by multiple components.

The IntelliButton uses the event information (passed in the form of a structured message) to find any matches; i.e. any components registered with the Locator that can consume the event. It uses simple type checking to identify potential services that the user may wish to call upon to operate on the data associated with the event. The matches are displayed to the user via the ActOn Button Bar, from which the user can select any or none of the integrating services suggested. If the user does choose one of the integrating services, the IntelliButton is notified and it accesses the correct service passing the associated data and event as parameters. In the above scenario, when the user highlighted the e-mail address, the IntelliButton used that event information to determine what services were available (send an e-mail, save the contact information, etc.) and suggested them.

ActOn Button Bar

The ActOn Button Bar, as described before, is simply the user interface for the integrating IntelliButton. We chose to keep the interface separate from the actual integrating functionality to allow easier experimentation with alternative interfaces. Currently, the interface is very simplistic. It is a dynamically generated list of buttons, where each button corresponds to a particular service that can be executed based on an user-generated event and its corresponding data. The list of buttons is provided by the IntelliButton. Each button is labeled with a short textual description of the following form:

<action> <datatype> using <service>

For example:

Send e-mail to this EmailAddress using Netscape.

Search for a string on the Web using AltaVista.

The ActOn button bar also provides short help messages when the mouse is placed over the button. These messages are provided by the individual service and are made available via the IntelliButton.

Desktop and Network Services

The previous three components discussed provide the core functionality of CyberDesk. Regardless of what tools the user wants to use, these three components are required. The fourth type of component, desktop and network services, are the actual services the user wants to access. Desktop services include e-mail browsers, contact managers, and schedulers. Network services include web search engines, telephone directories, and map retrieval tools.

To be included into the CyberDesk system, these services must register themselves with the Locator, providing a component handle and a list of interfaces that they support. These interfaces declare the list of services that they can be called upon to provide, and a set of data selection events that they can generate that could be used to trigger integrating behaviour. Currently, most data selection events are generated when the user selects some text with the mouse. Others are generated when significant changes in status occur, as will be seen in the section on higher level context.

The declaration implementation is usually a simple matter of writing a wrapper object for an existing service. Currently, the wrapper must be written by either the service designer, end user, or a middleman. We are looking at ways to automate this process. One method is to force all components in the system to support a common interface, like the JavaBeans initiative. This would enable the CyberDesk system to query each component and determine the events it can consume and generate.

One of the services available in CyberDesk is a gateway to the AltaVista search engine available on the web. The wrapper for this service, that allows it to interact with other CyberDesk components, consists of two main pieces. The first piece handles the declaration of its "method" interface to the Locator, stating that it can perform a web search on a String:

```
CameoProperty properties =  new CameoProperty
        ("searchFor", Class.forName("java.lang.String"),
        "Search for a string on the Web using Altavista");
CameoInterface interfaces =
        new CameoInterface("method", properties);
```

The second piece actually implements the search when called upon by the IntelliButton. With this interface, this search would be suggested by the IntelliButton whenever a text string is the target of a selection (assuming the component in which the text selection is done, supports the "select" interface). By their very nature, none of the network services support the "select" interface. They usually can not generate events and are of the form: receive input data and display output data. However, we will see how we can exploit this to provide even more interesting integrating behaviour in a process called "chaining".

The desktop services are a little more complicated because they have the potential to support the "select" interface. This means the wrapper has to deal with generating the necessary data selection events. In this case, the wrapper has an interface declaration section, as before, where it declares any "method" and "select" interfaces. For example, the Scheduler's interface is:

```
CameoProperty properties =  new CameoProperty
        ("lookupDate",
        Class.forName("cyberdesk.types.Date"),
        "Goto the date in the Scheduler");
CameoInterface interfaces[0] =
        new CameoInterface("method", properties);
CameoInterface interfaces[1] =
        new CameoInterface("select", null);
```

The first interface declares that it can consume date selection events and the second interface declares that it can generate data selection events.

The second section, where it implements the interfaces is slightly more complicated than with the network services. The wrapper must have "hooks" into the original application code to intercept and broadcast the appropriate data selection events (for the "select" interfaces), and to execute a service on data passed to it (for the "method" interfaces).

At the time of development, there were three ways to approach this problem for the "select" interface. First, we could modify the original application's event processing loop to broadcast events in the CyberDesk fashion. Second, we could modify the original application code to make calls to a notification routine in the wrapper when data is selected. Third, we could rely on the original application to have a suitable API for retrieving those events. Obviously the third method is the simplest and is not intrusive to the original application. Unfortunately, not all of the applications had APIs that allowed us to retrieve the necessary data selection events.

All of the desktop applets currently being used in CyberDesk (2 e-mail browsers, contact manager, 2 calendar managers/schedulers, scratchpad) were previously written by other Georgia Tech students. For those that did not provide sufficient APIs, we used the second method for capturing data selection events. It was far less intrusive than the first method, and we had access to the original code, allowing us to make changes.

In the newest release of the Java Development Kit (version 1.1), support was added for transferring data between (Java and non-Java) applications via a clipboard-style interface[7]. The use of this feature will allow us to avoid altering any application code in future versions of CyberDesk.

The problem is much simpler for the "method" interface. Either the application contained a method for acting on the given data, or it didn't. In cases where it didn't, we added additional methods to act on provided data. Note, that this didn't change the fundamental integration behaviour of CyberDesk, but only added additional features for us to exploit.

Type Converters

Data typing is used extensively in the interface declarations of the event sources and sinks that applications provide. The property field that corresponds to each interface declares the datatype/event that a component is interested in or can provide. The CyberDesk system takes advantage of the Java type system to do the data typing.

Initially, we hardcoded applications to generate events for different data types. For example, the e-mail browser declares that it can generate String selection events when text is highlighted, but also EmailAddress selection events when the "To:" or "From:" field in an e-mail message is selected. When EmailAddress selection events were generated, they were passed through the CyberDesk system, as described before, to the ActOn Button Bar, which displayed services that could consume EmailAddress selection events (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, we chose 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 our e-mail browser with this capability, so that any time text was selected in it, it would try to convert the text to an EmailAddress object and create an EmailAddress selection event rather than just a String selection event. But, rather than just giving this type conversion capability to the e-mail browser, we wanted to add that ability to the system once, and allow it to be used in every application where e-mail addresses might appear. We took the type detection ability out of the individual applications and created type converters, an independent and extensible layer in the architecture.

CyberDesk type converters behave a lot like the IntelliButton. When new components are added to the system, the converters determine which ones they are interested in, so they can add themselves to their list of observers. For example, the StringToEmailAddress converter is interested in all components that support the "select" interface for String objects and wants to observe them. So, when any component generates a String selection event, the StringToEmailAddress converter (and any other observers) are notified, and the converter attempts to convert the given String object to an EmailAddress object (while other converters attempt to convert the object to another CyberDesk type). In the above scenario, this conversion was done when the user selected the e-mail address. The system initially saw the selected data as a String but with this converter, it also saw it as an EmailAddress. This results in two related data selection events to arrive at the IntelliButton: one containing a string and one containing an EmailAddress. The IntelliButton will therefore seek integrating behaviour for both these types, allowing the user to access EmailAddress-relevant services where originally they wouldn't have had the option.

Currently the list of CyberDesk types include: Date, PhoneNumber, MailingAddress, Name, URL, and EmailAddress. If any of the conversions can be made, then the converter generates a second, but related, selection event containing the newly typed data and sends it to observing entities. This data also contains the original event. The system uses this information to ensure the type converters do not create an infinite loop (e.g. StringToEmailAddress, EmailAddressToString, StringToEmailAddress, etc.).

WHAT DOES CYBERDESK GAIN US?

CyberDesk provides a simple framework for adding new services and integrating them in reasonably intelligent ways. It relieves burdens from both the individual service designer and the end user. The individual service designer can develop a generic service, with a usable API, and not have to worry how it will be integrated into CyberDesk. The designer does not have to design specifically for the CyberDesk framework. The designer also doesn't have to think of all possible ways a user may want to integrate this service with another service, because the integrating behaviour is inherent to the CyberDesk framework. CyberDesk creates a dynamic mapping at runtime from user actions to possible user actions, saving the designer from constructing this map at design time.

CyberDesk makes things easier for the user as well. The user has the ability to easily add and remove services from the framework and does not need to hunt for ways to integrate various tools. The user is often supplied with integrating suggestions that they do not expect or had not thought of, but are appealing nonetheless.

EXTENSIONS

The CyberDesk framework was designed to be easily extensible. Simple extensions to CyberDesk include adding additional types, type converters, desktop services and network services. The real advantages with CyberDesk can be seen with more complex extensions that include adapting the behaviour of CyberDesk to individual use and creating more interesting integrating behaviour.

Adding Types and Type Converters

Type converters and types are often designed together, because they are intrinsically connected. To make the task easy for a designer, we have implemented a ConversionApplet class which handles all the CyberDesk communications and functionality. The designer is just required to implement three abstract methods:

weCanConvert(selected_data)
: determines if the input data is a type the converter can use

potentialLoop(original data selection event and related selection events)
: determines if this data was already converted to this type, checking for infinite loops

tryToConvert(selected_data)
: code that actually tries to convert the data to the output object of the type converter

An example converter is the StringToEmailAddress converter, which is a subclass of the ConversionApplet class. The code for this component and all components described in the paper can be viewed at http://www.cc.gatech.edu/fce/cyberdesk/samples. This converter looks at traditional ways of writing an e-mail address, and tries to map selected data to one of these ways. If it is successful, it returns an EmailAddress object. The ConversionApplet object is responsible for handling the ties to the CyberDesk framework.

Adding Desktop Services

As stated before, desktop services are a little more difficult to implement than other components. The main reason is that desktop services tend to have more complex functionality than network services or type converters. Once a desktop application has been written, it is fairly straightforward to implement a wrapper for use with CyberDesk. The wrapper must be a subclass of the original desktop application (slightly more difficult if the application is not written in Java, but is possible using the Java Native Interface [8]). The example below is the wrapper for the Contact Manager (see Figure 3), and it extends the ContactApplet class (the original application class).

```
public class ContactManager extends ContactApplet
implements CameoObject, Observer {
```

The wrapper must declare its interfaces,

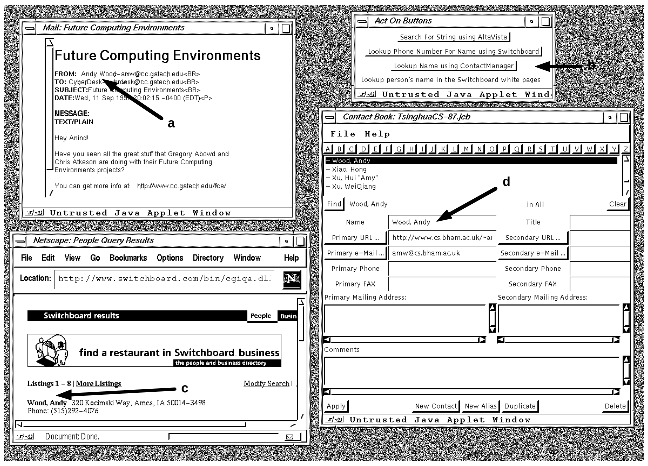
```
CameoInterface[] interfaces =
        new CameoInterface[2];
interfaces[0] = new CameoInterface("select", null);
CameoProperty[] properties =
        new CameoProperty[1];
properties[ 0 ] =
        new CameoProperty("lookup",
        Class.forName("cyberdesk.types.Name
        "Lookup an entry for the name in the
                ContactManager" );
interfaces[1] =
        new CameoInterface("method", properties);
```

provide methods to execute any services it provides,

```
/* method for invoking services supported by Contact
Manager */
public void manipulate(CameoMessage msg) {
        if ( msg.getField
        (msg.PROPERTY).equals("lookup")) {
                String name = ( (Name)msg.getField
                        (msg.DATA)).getName();
                /* call original ContactApplet method */
                showMe( name );
        }
}
```

and provide a way to generate data selection events.

The last requirement is a method that should be called whenever data is selected in the application. Generating these calls would often require us to intrude on the application that we're wrapping, however we have defined a simple selection API that if implemented by the application, circumvents this somewhat. The selection API was based on the Observer interface (note the class declaration above implemented the Observer interface) and Observable class provided in the Java language (version 1.0). It allows the designation of an object as Observable. Any object can choose to observe changes in an Observable object. For CyberDesk's purposes, an application must designate the data selected by a user to be observable and when the selected data changes, it must notify all observers.


Mail: Future Computing Environments
Untrusted Java Applet Window
a
Act On Buttons
Search For String using AltaVista
Lookup Phone Number For Name using Switchboard
Lookup Name using ContactManager
b
Netscape: People Query Results
File Edit View Go Bookmarks Options Directory Window Help
Location:
Switchboard results
People
Busin
find a restaurant in Switchboard business
the people and business directory
Listings 1 – 8 | More Listings
Modify Search |
c
Document: Done.
Contact Book: TsinghuaCS-87.jcb
File Help
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
Find
in All
Clear
d
Name
Title
Primary URL ...
Secondary URL ...
Primary e-Mail ...
Secondary e-Mail ...
Primary Phone
Secondary Phone
Primary FAX
Secondary FAX
Primary Mailing Address:
Secondary Mailing Address:
Comments
Apply
New Contact
New Alias
Duplicate
Delete

Future Computing Environments

FROM: Andy Wood - amw@cc.gatech.edu<BR>
TO: CyberDesk - cybrdesk@cc.gatech.edu<BR>
SUBJECT:Future Computing Environments<BR>
DATE:Wed, 11 Sep 1996 20:02:15 -0400 (EDT)<P>

MESSAGE:
TEXT/PLAIN

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/

Lookup person's name in the Switchboard white pages

http://www.switchboard.com/bin/cgiqa.dl

Wood, Andy 320 Kocimski Way, Ames, IA 50014-3498
Phone: (515)292-4076

- Wood, Andy
- Xiao, Hong
- Xu, Hui "Amy"
- Xu, WeiQiang

Find: Wood, Andy in All

Name: Wood, Andy
Primary URL ...: http://www.cs.bham.ac.uk/~ar
Primary e-Mail ...: amw@cs.bham.ac.uk

Figure 3: Screenshot of contact manager being used with CyberDesk. The user selects the string "Andy Wood" in the e-mail tool (a). CyberDesk offers some suggestions (b): search using Altavista, look up a phone number using Switchboard (c), and look up the name in the contact manager (d).

Adding Network Services

Much like the ConversionApplet, we have a ServiceApplet which is responsible for handling CyberDesk functionality and communications for network services. The ServiceApplet defines a set of methods which a service must implement:

initInterfaces()

declares the interfaces the service supports

manipulate(message containing selected data)

performs service(s) on the input data

The following is an example network service: WhoWhereEmail. It is a gateway to the WhoWhere network service available on the web. When a name is input into the web service, a list of possible e-mail addresses corresponding to that name is returned. The CyberDesk network service takes a Name object, inputs it into the service and displays the results in a web browser. The complete class definition is in Appendix B, with pertinent selections below. The class declaration is a subclass of the ServiceApplet.

```
public class WhoWhereEmail extends ServiceApplet {
```

The service declares its interfaces as follows:

```
properties[ 0 ] =  new  CameoProperty(
        "LookupEmailAddressFor",
        Class.forName( "cyberdesk.types.Name" ),
        "Lookup person's e-mail address in the
                WhoWhere listings");
return new  CameoInterface( "method", properties );
```

The actions the service can perform are defined in the manipulate method:

```
/* make sure the property is correct */
if (msg.getField(msg.PROPERTY).
equals("LookupEmailAddressFor")) {
        /* convert Name to usable form and
construct URL */
        String name = ( (Name)msg.getField(
                msg.DATA ) ).getName();
        URL search = new URL(new String
("http://query1.whowhere.com/jwz/name.wsrch?"
                + "name="+name));
        /* load URL and display in browser */
        getAppletContext().showDocument(
                search, "_whowhere" );
}
```

This is a simple service, interested in only one data selection event - a Name. Other services are more complex and are interested in multiple selection events, but they are written and work in the same way. They declare additional properties, and implement the service for that property in the manipulate method.

We have had several students develop simple extensions for CyberDesk, including desktop and network services, type converters and types. As a testament to the ease of development, at last count there were 6 desktop services, 68 network services, 7 type converters, and 6 data types.

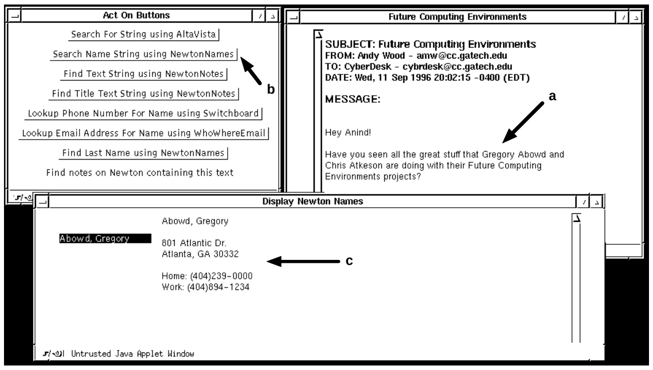
Case Study: Accessing Mobile Data

A project currently under development in our research group is LlamaShare, an architecture and set of applications for providing users and programmers easy access to information stored on mobile devices. There are two main goals for the LlamaShare project, one of them coinciding with a goal of CyberDesk. The first is to create an infrastructure that makes it simple for programmers to take advantage of mobile data in their applications. The second is to provide applications that demonstrate ubiquitous access to information.

Currently, it is very difficult to get information off of a mobile device (a PDA like a Newton, for example) both for programmers and for users. From a user's perspective, it is also very difficult to deal with information stored on a mobile device. The current method of accessing this data is typically through a "synchronization" process, which does a reasonable job of copying the data to a user's desktop machine, but does nothing to aid them in actually doing anything with that information, such as integrating relevant pieces into their daily tasks. The LlamaShare infrastructure, consisting of a central server called the LlamaServer, provides routing for information requests between any mobile device on the network (wired or wireless) and any desktop machine on the Internet.

There were two reasons for integrating CyberDesk with LlamaShare. First, we wanted to illustrate the platform-neutrality and language-neutrality of the LlamaServer, which CyberDesk allows us to do. More importantly, however, CyberDesk's vision of ubiquitous information access was the deciding factor. While LlamaShare provides a concrete, visible object to represent the data on a mobile device, CyberDesk takes the approach that information is distributed throughout a rather nebulous information space (consisting of Internet, desktop, and mobile data) that can be retrieved at any moment depending on the context in which the user is currently working. This new metaphor of seamless integration between mobile data and Internet (remote) data was too good to pass up.

Adding services and viewers to CyberDesk was quite simple. The most common data types that users would be interested in on their PDA (text, names, phone numbers, and dates) were already supported by CyberDesk so nothing new needed to be added.



Act On Buttons
Search For String using AltaVista
Search Name String using NewtonNames
Find Text String using NewtonNotes
Find Title Text String using NewtonNotes
Lookup Phone Number For Name using Switchboard
Lookup Email Address For Name using WhoWhereEmail
Find Last Name using NewtonNames
Find notes on Newton containing this text
b
Future Computing Environments
a
Display Newton Names
c
Untrusted Java Applet Window

SUBJECT: Future Computing Environments
FROM: Andy Wood - amw@cc.gatech.edu
TO: CyberDesk - cybrdesk@cc.gatech.edu
DATE: Wed, 11 Sep 1996 20:02:15 -0400 (EDT)

MESSAGE:

Hey Anind!

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

Abowd, Gregory

Abowd, Gregory

801 Atlantic Dr.
Atlanta, GA 30332

Home: (404)239-0000
Work: (404)894-1234

Figure 4: Screenshot of LlamaShare being used in CyberDesk. The user selects a name (a) in the e-mail tool. CyberDesk offers a number of integrating suggestions (b), including 4 that access data from a remote Newton. The user chooses the second suggestion and sees the results (c), obtained from a remote Newton.

The next task involved adding the services that recognized the appropriate types and created "ActOn" buttons for them. We added two services (NewtonNames and NewtonNotes) which, respectively, request contact information from the Newton about the selected name and request notes from the Newton containing the selected text in the body or title. Adding these services was quite simple (see Figure 4), requiring only the implementation of the ServiceApplet methods described above.

The code that displays the results of the query to the Newton is stored in the NewtonDisplayNames class which knows nothing about CyberDesk at all. It is just a class that uses Java libraries to create a window to display information and can be used from either an applet or an application.

Here are some examples of what we are doing with LlamaShare and CyberDesk:

- You're writing some e-mail and you know you scribbled down a note on your Newton with some relevant text in it. Select text and search for all the notes on your Newton containing that text, then read them on your desktop machine.
- You get e-mail from a colleague and you want to call them back. You have her phone number on your Newton. Select her name to pull her name card from your Newton and display it on your workstation.
- Your secretary took down the number of someone who called while you were in a meeting, but you don't know who that number belongs to! Select the phone number and let the Newton search its names database and return the correct person or company matching that phone number.
- Your boss sends you e-mail asking you to schedule a meeting with two other people in your group. How do you find a time that everyone can meet? Select each of their names and let CyberDesk pull up their calendar's from their respective Newtons and display them. Schedule the meeting and make the change effective immediately in their Newton calendar.

Chaining

The extensions described so far are fairly simplistic. They deal with adding more suggestions for the user. More interesting are extensions which add more powerful suggestions for the user. Chaining is an example of this type of extension. It extends CyberDesk's type converting ability by using network services as type converters, to allow for increased integrating behaviour.

For example, a user is reading an appointment in her calendar manager, and selects the name of the person she's supposed to be meeting. 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 WWW, etc. However, by using chaining, more powerful suggestions can be had.

The WhoWhereEmail network service described earlier, takes a name as input and returns a WWW 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 selection event, and the user is supplied with all possible suggestions for both a Name and an EmailAddress.

Designing this ability was very simple. We just had to create a class of objects that supported all the attributes of a ServiceApplet and a ConversionApplet. All network services already implement the ServiceApplet requirements, so they just had to be modified to support the three methods that the ConversionApplet required.

This ability is very powerful for the user, providing another dimension of suggestions for each type the selected data can be converted to. It allows us to support interactions like those described in the hypothetical user scenario: the user was at his friend's house, and the system offered him suggestions on calling or e-mailing his friend, or looking at his friend's calendar. This chaining example converts a Location object to a Name, EmailAddress, PhoneNumber and Schedule objects.

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 services, both network and desktop, should be able to take advantage of these multiple types. They can, through a process we call combining.

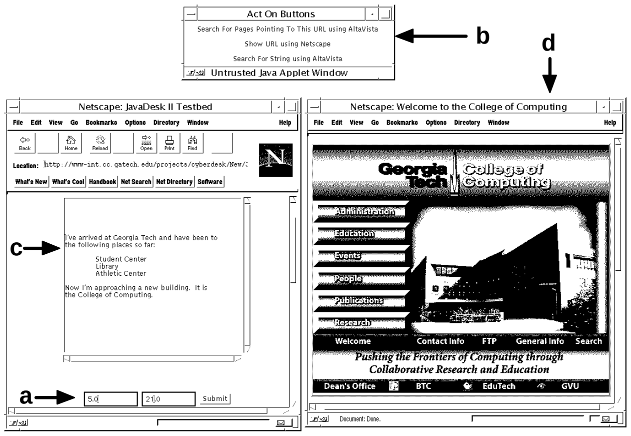
Combining, in CyberDesk terms, is the ability to collect multiple data types for a piece of selected data, and bind them together, as needed, to create multiple meta-objects. These meta-objects could be used to perform substantially more powerful actions. Taking the above example of a user reading an appointment in her calendar, assume that there are multiple services capable of chaining, so a URL selection event is created, a Date selection event is created, a MailingAddress selection event is created, etc. There is potentially enough information to create a new entry in the Contact Manager. The ability to assimilate this widespread information into a compact entity is very powerful for a user. It's this same ability that would allow us to combine time and a name to search a friend's schedule as seen in the user scenario. An example of a combining extension we've created is shown below.

Higher Level Context

CyberDesk contains some simple notions of context. It knows the application a user is working with and the data (both type and content) the user is interested in (via explicit selection with the mouse). But CyberDesk has shown the potential for supporting higher level 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 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, behaviour, etc. While there has been a lot of research in context-aware applications [13,9,14], we are not aware of a general toolkit which supports higher level context and integration behaviour like CyberDesk does.

We have implemented prototype services that accept position and time information for use with CyberDesk. Ideally, the position service would notify the system when there was a significant change in position, where "significant" depends on the user context. For example, in the user scenario, significant position changes events were generated when the user moved from one building to another. If a user is inside a building, a significant change could be a move from one room to another. This could be accomplished by having a single service that can use multiple levels of detail or multiple services each responsible for a separate level of detail.


Act On Buttons
Search For Pages Pointing To This URL using AltaVista
Show URL using Netscape
Search For String using AltaVista
Untrusted Java Applet Window
b
d
Netscape: JavaDesk II Testbed
File Edit View Go Bookmarks Options Directory Window Help
Back Home Reload Open Print Find
Location: http://www-int.cc.gatech.edu/projects/cyberdesk/New/:
What's New What's Cool Handbook Net Search Net Directory Software
c
a
5.0
21.0
Submit
Netscape: Welcome to the College of Computing
File Edit View Go Bookmarks Options Directory Window Help
Georgia Tech College of Computing
Administration
Education
Events
People
Publications
Research
Welcome Contact Info FTP General Info Search
Pushing the Frontiers of Computing through Collaborative Research and Education
Dean's Office BTC EduTech GVU
Document: Done.

I've arrived at Georgia Tech and have been to the following places so far:

Student Center
Library
Athletic Center

Now I'm approaching a new building. It is the College of Computing.

Figure 5: 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).

The prototype position service we've developed is for the campus of Georgia Tech. The service coarsely maps GPS coordinates to buildings on campus. Figure 5 is a screenshot showing this service being used. When the updated coordinates correspond to a new building, a Location selection event is generated, which is then used in combination with the rest of the CyberDesk system. We would like to combine this service with a knowledge of a user's history. So, when a user approaches a building they've never been to before, the CyberDesk system should offer introductory information on the building. If the user has been to the building before, different sets of information should be offered.

Our prototype time service is currently hardcoded to announce Time change events every few minutes. When combined with a Calendar Manager, the behaviour is that of a reminder service for scheduled events. We are currently looking at more flexible methods of implementing this. We also use time in combination with names, to determine the best way to contact people, as shown in the sample user scenario.

While we have not developed all the services to fully realize the user scenario given at the beginning of the paper, we have demonstrated and developed services that will enable us to do so shortly.

CURRENT LIMITATIONS

The CyberDesk framework was designed to be easily extensible and easy to use, however it suffers from a few limitations. To ease implementation and to support ubiquitous access, we chose a web browser as our runtime environment. Current browsers (with implementations on several platforms) have security managers limiting what Java applets have access to. In general, applets can only access network information, and not local information. This limits the scope of applications that can be used in the CyberDesk framework. As well, most web browsers have Java Virtual Machines (JVMs - interpreters that execute Java byte code) that are severely limited in functionality. We have hit the limit on the number of applets/components that can be running at any one time. The exact number depends on the types of applets running, and in our case, the number appears to be approximately 15. We feel this number is too low to see the powerful ability of chaining and combining.

To solve these problems, we have built a version of CyberDesk that runs outside the browser. It still allows access to all network services and desktop services, and WWW browsers. The components still share a single JVM, but it is more powerful than ones typically found in browsers, allowing access to local information and allowing a greater number of components to run simultaneously. This prototype is currently undergoing testing.

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 long list of suggested user actions. It is clear that the number of possible suggestions could quickly become overwhelming to the user. We are currently looking at different ways to adapt the interface to initially show actions that the user is likely to take, but provide 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 and document lenses [4].

One of the problems we've found with chaining is that there is the potential for multiple services to generate a data type: a Name object, for example. Since the services are running independently, the Name objects that they generate could be different. If a suggested action to the user is to put this name in the Contact Manager, which Name object should be used? We need to investigate methods for determining relevancy and confidence of suggested actions and results, in order to rank suggestions for the user.

RELATED WORK

Pandit and Kalbag's Selection Recognition Agent [12] attempts to address the same issues as CyberDesk. Unlike CyberDesk, it uses a fixed datatype-action pair, allowing for only one possible action for each datatype 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.

Apple Data Detectors [2] is another component architecture that supports automatic integration of tools. It works at the operating system level, using the selection mechanism that most Apple applications support. It allows the selection of a large area of text and recognizes all user-registered datatypes in that selection. Users view suggested actions by pressing a modifier key and the mouse button. Like CyberDesk, it supports an arbitrary number of actions for each datatype. It does not support chaining and supports only a very limited notion of combining. When a datatype is chosen, a service can collect related information and use it, but this collected information (CyberDesk's meta-object) is not made available to other services. The Apple Data Detectors system does not support the use of higher level user context, such as position. Its focus appears to be desktop applications, as opposed to CyberDesk's ubiquitous services, existing either locally or remotely.

FUTURE WORK

We will continue to add desktop and network 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 chaining and combining
- searching for other advanced techniques, like chaining and combining
- investigating learning-by-example techniques [5] to allow the CyberDesk system to dynamically create chained suggestions based on a user's repeated actions
- incorporating rich data types into CyberDesk, other than time, position, and meta-types. This will allow us to use CyberDesk as the platform for developing context-aware applications.

- experimenting with adaptive interfaces and different interface representations in order to determine better ways of presenting suggestions to our users.
- automating the registration process of components, so that no one will be required to write the component wrappers we use now.

CONCLUSIONS

CyberDesk is a component software framework that provides automatic integration between components: desktop and network services. The automatic integration is a result of the dynamic mapping performed at runtime, between user actions and possible user actions. By performing this mapping at runtime instead of at design time, as is traditionally done, we move the difficult design decisions out of the designers' hands, and provide complete flexibility and customizability to the user. With the basic CyberDesk system, a number of interesting integration suggestions are offered to the user. With the use of advanced extensions like chaining, combining, and the use of higher level context, more powerful integration suggestions can be obtained. These advanced extensions are possible without any modifications to the basic CyberDesk structure. In the future, we plan to investigate the user of other advanced extensions and study our current extensions further. We will also look at advanced techniques for automating the development of CyberDesk components and alternative user interfaces.

ACKNOWLEDGMENTS

We would like to thank Siddharth Bajaj, Jason Ellis, Dav Haas, George Riley, and Ashley Taylor who have developed components for CyberDesk. We would also like to thank Alan Dix, Dav Haas, and Scott Hudson for their invaluable comments on our work and this paper.

REFERENCES

1. Apple Computers. OpenDoc homepage. Available at http://www.opendoc.apple.com.
2. Apple Research Labs. Apple Data Detectors homepage. Available at http://www.research.apple.com/research/tech/AppleDataDetectors/.
3. Berwick, R. et al. Research Priorities for the World Wide Web. Report of the NSF Workshop Sponsored by the Information, Robotics, and Intelligent Systems Division. (Arlington, VA, October 31, 1994).
4. Bier, E.A. et al. ToolGlass and Magic Lenses: The See-Through Interface. Computer Graphics Proceedings, Annual Conference Series, 1993. ACM SIGGRAPH, New York. 73-80.
5. Cypher, A. EAGER: Programming repetitive tasks by example. In Proceedings of CHI' 91. ACM Press.
6. JavaSoft. JavaBeans homepage. Available at http://splash.javasoft.com/beans/.
7. JavaSoft. AWT Data Transfer homepage. Available at http://www.javasoft.com/products/jdk/1.1/docs/guide/awt/designspec/datatransfer.html.
8. JavaSoft. Java Native Interface homepage. Available at http://www.javasoft.com/products/jdk/1.1/docs/guide/jni/index.html.
9. Long, S. et al. CyberGuide: Prototyping Context-Aware Mobile Applications. In Proceedings of CHI '96 (Vancouver, Canada, March 1996), ACM Press.
10. Microsoft. ActiveX homepage. Available at http://www.microsoft.com/activex/.
11. Microsoft. OLE Development homepage. Available at http://www.microsoft.com/oledev/.
12. 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.
13. Schilit, B. A Context-Aware System Architecture for Mobile Distributed Computing. Ph.D. Thesis, Columbia University, May 1995.
14. Want, R. et al. An Overview of the PARCTAB Ubiquitous Computing Experiment. IEEE Personal Communications 2 (6). 1995. 28-43.
15. Wood, A. CAMEO: Supporting Observable APIs. Position Paper for the WWW5 Programming the Web Workshop. (Paris, France, May, 1996).
16. Wood, A., Dey, A., Abowd, G. CyberDesk: Automated Integration of Desktop and Network Services. Technical Note In Proceedings of CHI' 97 (Atlanta, GA, March 1997), ACM Press.