EXHIBIT 22

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.

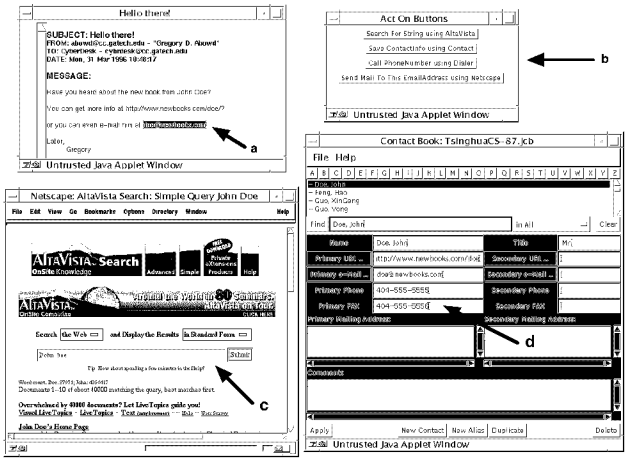

Figure 1. Mock screenshot of above user scenario

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.

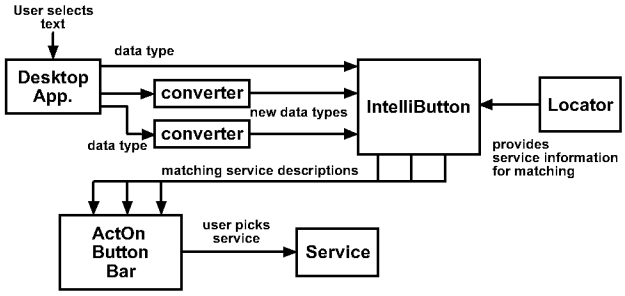

Figure 2: Runtime architecture diagram

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.

DOCKET ALARM

Explore Litigation Insights

Docket Alarm provides insights to develop a more informed litigation strategy and the peace of mind of knowing you're on top of things.

Real-Time Litigation Alerts

Keep your litigation team up-to-date with **real-time alerts** and advanced team management tools built for the enterprise, all while greatly reducing PACER spend.

Our comprehensive service means we can handle Federal, State, and Administrative courts across the country.

Advanced Docket Research

With over 230 million records, Docket Alarm's cloud-native docket research platform finds what other services can't. Coverage includes Federal, State, plus PTAB, TTAB, ITC and NLRB decisions, all in one place.

Identify arguments that have been successful in the past with full text, pinpoint searching. Link to case law cited within any court document via Fastcase.

Analytics At Your Fingertips

Learn what happened the last time a particular judge, opposing counsel or company faced cases similar to yours.

Advanced out-of-the-box PTAB and TTAB analytics are always at your fingertips.

API

Docket Alarm offers a powerful API (application programming interface) to developers that want to integrate case filings into their apps.

LAW FIRMS
Build custom dashboards for your attorneys and clients with live data direct from the court.

Automate many repetitive legal tasks like conflict checks, document management, and marketing.

FINANCIAL INSTITUTIONS
Litigation and bankruptcy checks for companies and debtors.

E-DISCOVERY AND LEGAL VENDORS
Sync your system to PACER to automate legal marketing.