Dartmouth College

Dartmouth Digital Commons

5-1-2000

An Infrastructure for a Mobile-Agent System that Provides Personalized Services to Mobile Devices

Debbie O. Chyi
Dartmouth College

Recommended Citation

Dartmouth College Computer Science Technical Report TR2000-370

An Infrastructure for a Mobile-Agent System that Provides

Personalized Services to Mobile Devices

Debbie Chyi
Dartmouth College
Department of Computer Science
May 2000

Advisor: David F. Kotz

Abstract

In this paper, we present the design of a mobile-agent system that provides a mobile user with a personalized information retrieval service and we describe the implementation of the infrastructure for such a system. This "Personal Agent System" gathers information from the Internet and uses context-aware mechanisms to manage the information according to a mobile user's needs and preferences. The user's schedule and location are the context indicators in this system. These indicators are critical in ensuring that users obtain only the information they want, receive information in a form that is most useful for viewing on their mobile device, and is notified of new information in a minimally intrusive manner. The system incorporates a rule-based learning system to enhance the personalization achieved by the system.

1. Introduction

With the wealth of information that has become available to users through distributed multimedia information services, including the World Wide Web, and with the advancement and increasing popularity of mobile devices, the information accessible to users is greater than ever before. Virtually any information is accessible any time and anywhere. The amount of information available, however, can easily overwhelm users and prevent them from fully utilizing their information resources. The need exists for an information-gathering system that is closely integrated with mobile devices and that personalizes its service to the user. This idea forms the basis of this project.

Using mobile agent technology developed at Dartmouth College, we designed a system that gathers information from the Internet and uses context-aware mechanisms to manage the information according to the user's needs and preferences. The user's schedule and location are the context indicators in this system. These indicators are critical in personalizing the information service so that users obtain only the information they want, receive information in a form that is most useful for viewing on their mobile device, and is notified of new information in a minimally intrusive manner. For this reason, our system focuses on personalizing services to suit the needs of a user and we call our system a "Personal Agent System".

The next section in this paper presents the background on this project. In Section 3, the paper discusses the design and architecture of the Personal Agent System. We describe the implementation of the system's infrastructure in Section 4. In Section 5, we discuss personalization of the system. Section 5.1 discusses the use of rules to represent user preferences and Section 5.2 describes both learning mechanisms that we chose for our rule-based system and learning mechanisms presented in other papers that may be useful to our system. Finally, we conclude the paper with a summary of our work on the Personal Agent System.

2 Background

Before we describe our system, we provide some background on mobile agents and on our hardware platform.

2.1 Mobile Agents

A mobile agent is a "named program that can migrate from machine to machine in a heterogeneous network" [4]. Mobile agents provide a convenient paradigm for working with communication channels and offer a reliable way for packaging programs and data to be sent and executed over the network. Because they are able to continue execution at each server that they migrate to, they are "pro-active and run autonomously in the network" [2]. The mobile agent system used for the project was *Pokegent*. *Pokegent*, a name that alludes to the phrase "pocket agent", supports agents running on a stationary host and on a mobile device. Specifically, it supports agents running on Windows NT and on Windows CE. This project uses the inter-communication capabilities of the agents heavily. Although we did not use the agents' migration capabilities, they will become critical when we extend the Personal Agent System.

The agents in the *Pokegent* system are written in C++. An agent server and a stationary "system agent" run on each machine that will host agents. Agents are dynamic-link libraries (DLLs) that are passed to the system agent upon creation and downloaded from the Internet upon migration. Essentially, agents are threads running in the environment of an agent server. An application program interface (API) for communicating between agents hides socket connections from the application programmer. The system of communication between agents is event-driven. Each agent has an event loop that processes incoming messages and the agent system automatically queues messages until they can be processed. Because the Personal Agent System relies so heavily on communication between a mobile device and a stationary host and its future

extensions will involve migrating processes through the network, we used mobile agents for the implementation of this project.

2.2 Platform

We developed the Personal Agent System on a Hewlett Packard Jornada (see Figure 1) running Windows CE 2.11, a Pentium II workstation running Windows NT 4.0, and a WaveLAN network for communicating wirelessly between the two machines. The Jornada used a Lucent WaveLAN IEEE 802.11 PC Card (Bronze) to communicate over the wireless network. Windows CE is an operating system that runs on mobile devices and supports multi-threaded programming. This support of multiple threads is a central reason why *Pokegent* was chosen to run on Windows CE. A multiple-threaded environment can better support a system of agents because the paradigm of an agent closely resembles that of a thread. Because Windows CE only supports Unicode and, of the Windows family, only Windows NT supports Unicode, Windows NT is the only other operating system that *Pokegent* supports.

Figure 1: Hewlett Packard Jornada

3. Design and Architecture

3.1 Overview

The Personal Agent System consists of agents, information servers, information-viewing applications, and context-related sensors (see Figure 2). The agents form the central part of the Personal Agent System. The actual Personal Agent resides on a stationary server and it communicates with agents residing on the mobile device. The agents in the system handle the gathering and personalization of information and communicate with one another extensively to provide an integrated service to the user. The information servers supply the Personal Agent with the various information that the agent requests. These information servers can reside on any machine and typically would be owned and maintained by third parties that wish to offer information services to users. For each form of information that the Personal Agent gathers, there is a corresponding user application that provides an interface for displaying and managing the information. Finally, the context-related sensors monitor the user's location and schedule and notify the appropriate agent when the user's context changes.
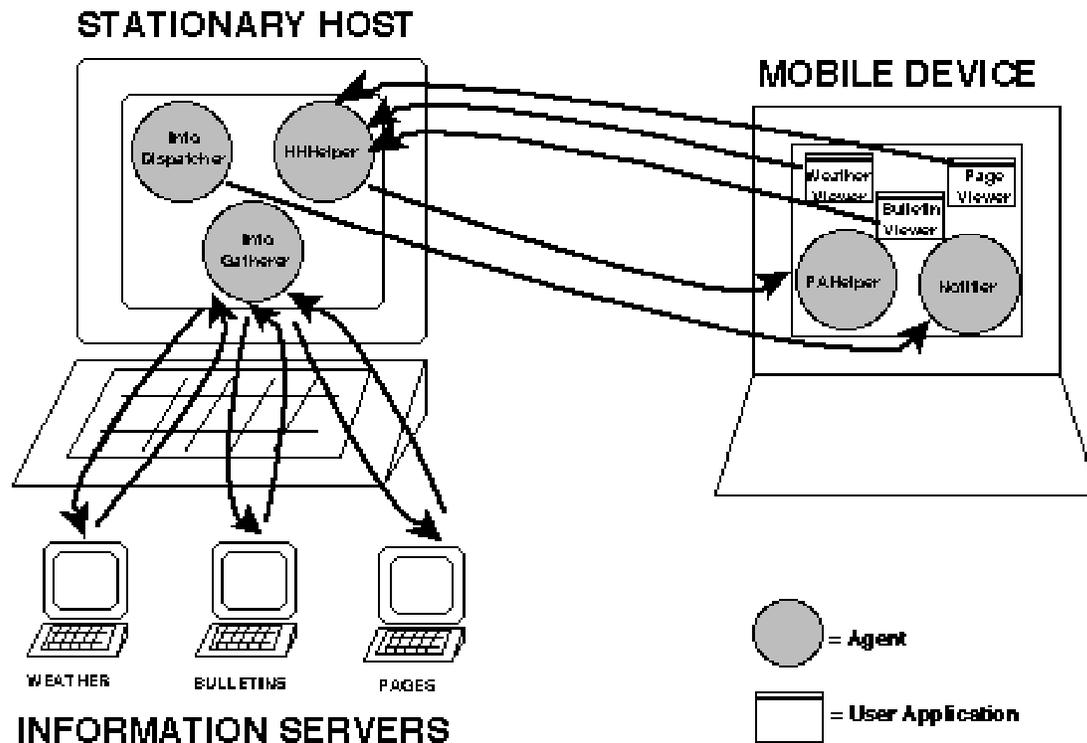
Figure 2: Overview of the Personal Agent System (Pages refer to one-way electronic messages)

The project involves personalizing information services by using the user's current context to filter and convert incoming information as well as to determine when to notify a user of new information. The project also involves migration of the Personal Agent to other stationary servers so that it follows the user around in the wired network while the user and her mobile device move around in the wireless network (see Figure 3). In other words, the Personal Agent always moves to a location that is close to the user. Having the Personal Agent and the agents on the mobile device in close proximity to each other (ideally in the same local network) allows communication between them to be fast and reliable.
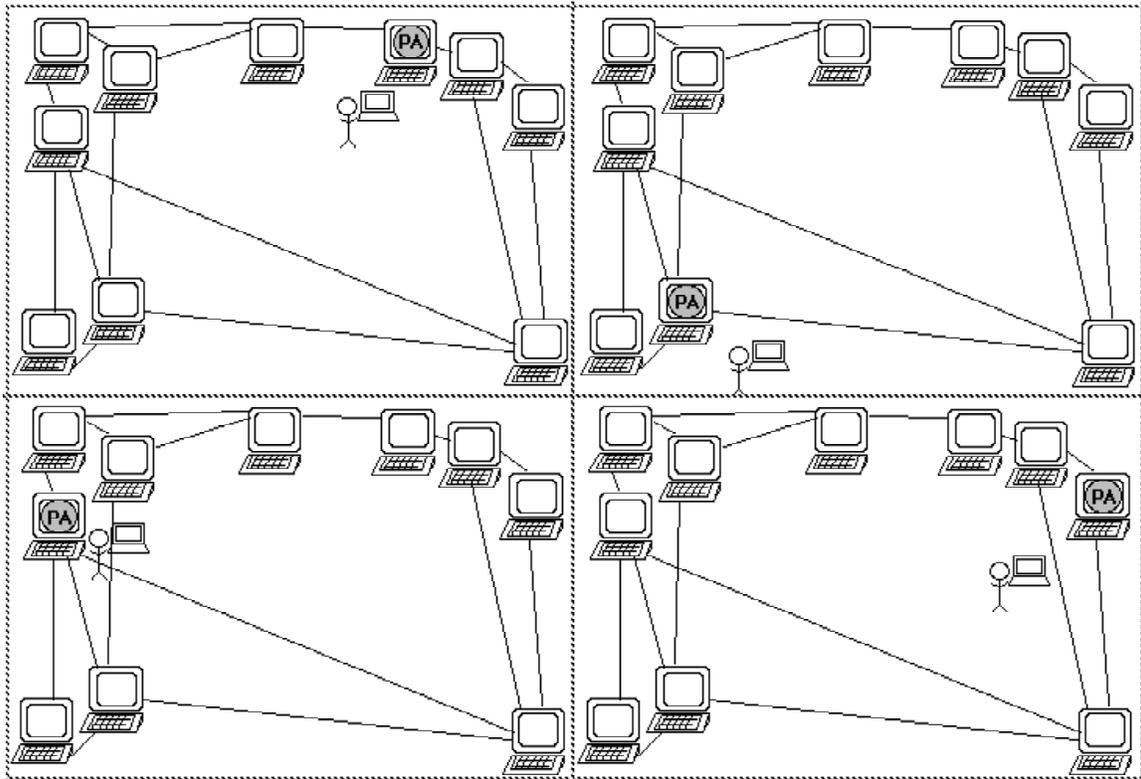
Figure 3: The Personal Agent (PA) always moves to a location that is close to the mobile user

A central concept in the personalization of our system is the concept of context-awareness. The system is aware of both the user's location and schedule. The general location of a user can be determined from the subnet in which he is located. The user's schedule can be determined from a calendar application that is on the mobile device and is updated by the user. When combined with user preferences, these two forms of information can be actively used to filter and convert incoming information as well as to determine the timing of notifications so that the user is presented with a minimally intrusive user interface. As mentioned in Chavez, Ide, and Kirste, the main goal of a mobile information system is to "provide the right information at the right time and

place—with minimal interaction" [3]. The article explains, "an optimal assistant provides the required information autonomously and independently, without requiring the user to ask for it explicitly" [3]. These ideas are central to the design of our project, too.

3.2 A Closer Look

The Personal Agent communicates with information servers over an ASCII socket and polls the information server at an interval specified by the user. The Information Gatherer thread within the Personal Agent opens a socket to an information server, sends the information request over the socket, and then receives the information over the socket. The information servers are stateless because the Personal Agent will constantly change information servers as it jumps from server to server. With each jump, the Personal Agent should not have to establish and delete its state on the information servers. An important aspect of the interaction between the Personal Agent and an information server is that the Personal Agent can provide location information as a command-line argument to its request. Since the Personal Agent is aware of what subnet it is in, for example, it can lookup its subnet in a web-based directory to determine the zip code corresponding to the subnet and then supply the zip code as a command-line argument to its request to the weather server. The weather server works with a weather site that provides zip-code-specific weather and therefore provides the Personal Agent with the local weather. Jose and Davies state that in "mobile distributed environments applications often need to dynamically obtain information that is relevant to their current location" but "the current design of the Internet does not provide any conceptual models for addressing this issue" [5]. We therefore address this issue by having a location-aware

agent supply its location to information servers that are programmed to obtain location-specific information from the Internet.

The agents that reside on the stationary host are the Information Gatherer, the Information Dispatcher, and the HHHelper (where HH refers to "Handheld"). These agents together comprise the actual Personal Agent. To avoid confusion, I refer to these three agents as threads of the Personal Agent. The Information Gatherer is the thread responsible for polling the information servers. The Information Dispatcher is the thread responsible for personalizing the data. The process of personalization includes deleting unwanted information, converting data into a desired form, and determining the details of user notification.

The Information Dispatcher extracts the appropriate personalization profile from the Personal Agent's collection of profiles and then uses the profile to determine how the incoming information should be processed. A personalization profile is simply a list of personalization rules and each information type has its own associated profile. If the personalization profile determines that the new information should be filtered out, then the information is deleted and the processing is completed. If however, the personalization profile allows the new information to filter through, then it will perform any needed conversion on the information and provide the Information Dispatcher with context data that specifies under what context the user should be notified of this new information and what message should be posted with the notification. If the network connection is down when it is trying to push data to the mobile device, the Information Dispatcher queues the data until a network connection is re-established.

The HHHelper is the thread within the Personal Agent that handles task requests from the mobile device. These tasks include changing preferences, carrying out cache management requests, and storing information about the user's current context.

The agents that reside on the mobile device are the Notifier agent and the PAHelper agent (where PA refers to "Personal Agent"). The Notifier receives new information pushed to the mobile device by the Personal Agent. When it receives new information, the Notifier provides the information to a user application and then uses its knowledge of the user's current context to determine if and when the user should be notified of the new information. Notifying the user, in most applications, entails displaying a message box with a brief summary of the new information. The PAHelper processes task requests from the Personal Agent. These tasks include receiving information files from the HHHelper, carrying out cache management requests, and storing information related to the location of the Personal Agent.

The user applications allow the user to view a window that lists entries for the existing information and to double-click on an entry to pop-up a window that displays the actual information. These applications also allow the user to manage existing information (i.e., delete information and set expiration times) and explicitly request new information. A user application exists for each type of information that the Personal Agent retrieves. These user applications interact with the Personal Agent System primarily through the Windows CE database. They do, however, also send events directly to agents in the system.

The context-related sensors include the Location Detector and the Calendar. These are Win32 applications--not agents. The Location Detector sends the mobile

device's IP address to the Notifier each time the user moves into a different subnet. The

Calendar application organizes the user's schedule and has a user interface that allows the

user to easily add, change, and delete activities. The user categorizes each activity

entered into the Calendar as one of the following types: Meeting, Class, Eating, or

Leisure. Users can also add their own categories. At the start of each activity, the

Calendar sends the activity's category to the Notifier. The system therefore uses these

categories to determine the user's schedule context. The more categories that the

Calendar uses the better informed the system is of the user's schedule and the better it can

adjust to the user's needs.

Ensuring a fast and reliable network connection between the Personal Agent and

mobile device (by having them in close proximity) is critical because the Personal Agent

continually pushes information to the mobile device, all requests from the mobile device

are routed through the Personal Agent, and the caching mechanism (below) requires

integrated data management between the Personal Agent and the mobile device.

Constant interaction between a mobile device and a stationary server allows users to take

advantage of the mobility offered by their mobile device as well as to take advantage of

the resources of the stationary server. These server resources include more memory,

faster processing, an unlimited power source, and a faster, more reliable network

connection.

Integrating a cache into the Personal Agent System allows the mobile device to

store frequently used data locally. We designed the cache as a three-level hierarchy

consisting of a local level, a Personal Agent level, and a "World" level. The data at the

local level is a subset of the data at the Personal Agent level while the data at the Personal

Agent level is a subset of the data at the "World" level. The "World" level is all information that can be obtained from any source available to the Personal Agent. With a data hierarchy, however, the user sometimes views out-of-date information. This situation occurs when the data that the user wishes to view exists on the local or Personal Agent level and is older than the "World" version. Unless the local version is deleted by the user or automatically replaced in the cache, the cache will never retrieve the newer version from the "World". Because not all data resides locally in this cache structure, a disconnected network causes additional complications. If the device tries to retrieve remote data but the network is disconnected, the system would have to notify the user that the data is currently unavailable.

4. Implementation

Because the focus of this project is to develop an infrastructure for the personalization of information services, we did not implement migration of the Personal Agent or caching. In our implementation, all data resides with the Personal Agent and we assume the Personal Agent has an unlimited storage capacity. This greatly simplifies data management because only a single version of any data ever exists. Although this storage decision does present the issue of a bottleneck and a single-point of failure, ideally it should not pose a serious problem since the connection between the Personal Agent and mobile device should be fast and reliable. For this section, the reader should refer to Figures 4 and 5 to gain a better understanding of the interaction between the various parts of the system.

The Personal Agent

Extracts personalization fields and saves the actual information to a file

Info Dispatcher

Information from Information Servers

New information's personalization fields and file path

Matches personalization fields to a rule in a personalization profile and obtains notification data from rule

Info Gatherer

HHHelper

User Application's request for contents of file

Opens file and reads in the contents

Contents of file to PAHelper

Display fields, file path, and notification data to Notifier

Figure 4: The Personal Agent

Displays listbox of information entries and displays
information when user double-clicks on entry

Request for
contents of
information file

Weather
Viewer

Bulletin
Viewer

Page
Viewer

Display fields, file path,
and notification data
from Info Dispatcher

Notifier

Places display fields and file path in CE
database and then handles notification
according to notification data (User
Application is automatically notified of new
information in database)

File contents requested
by User Application

PAHelper

Saves contents to CE database
(User Application is automatically
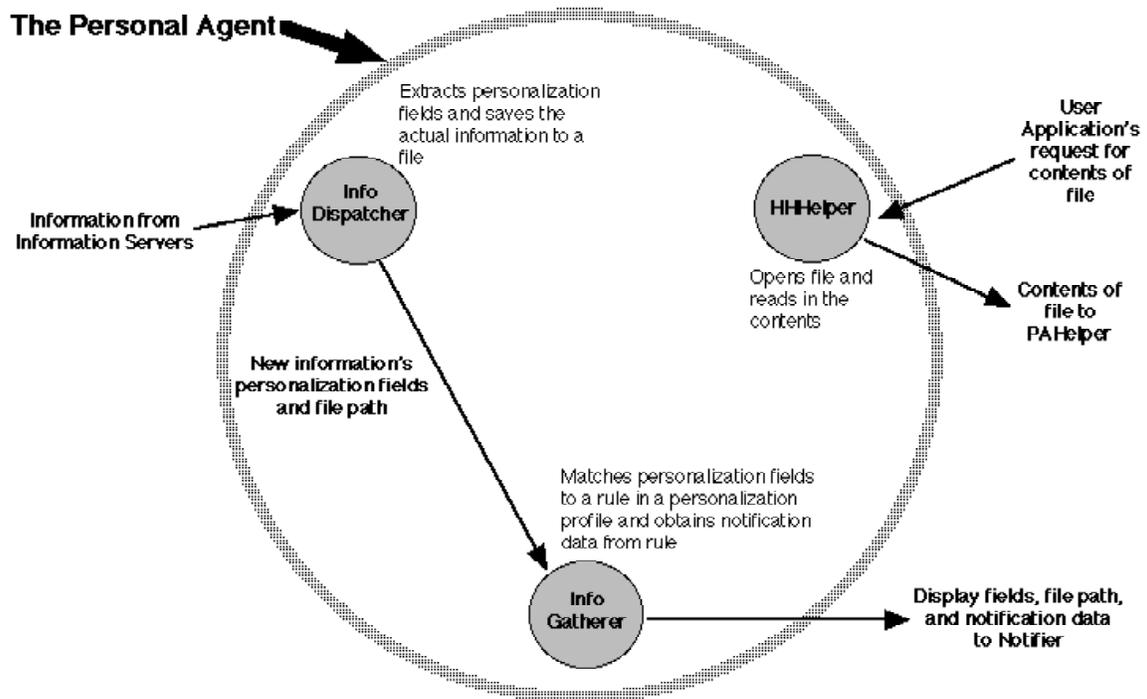notified of content's arrival in
database)

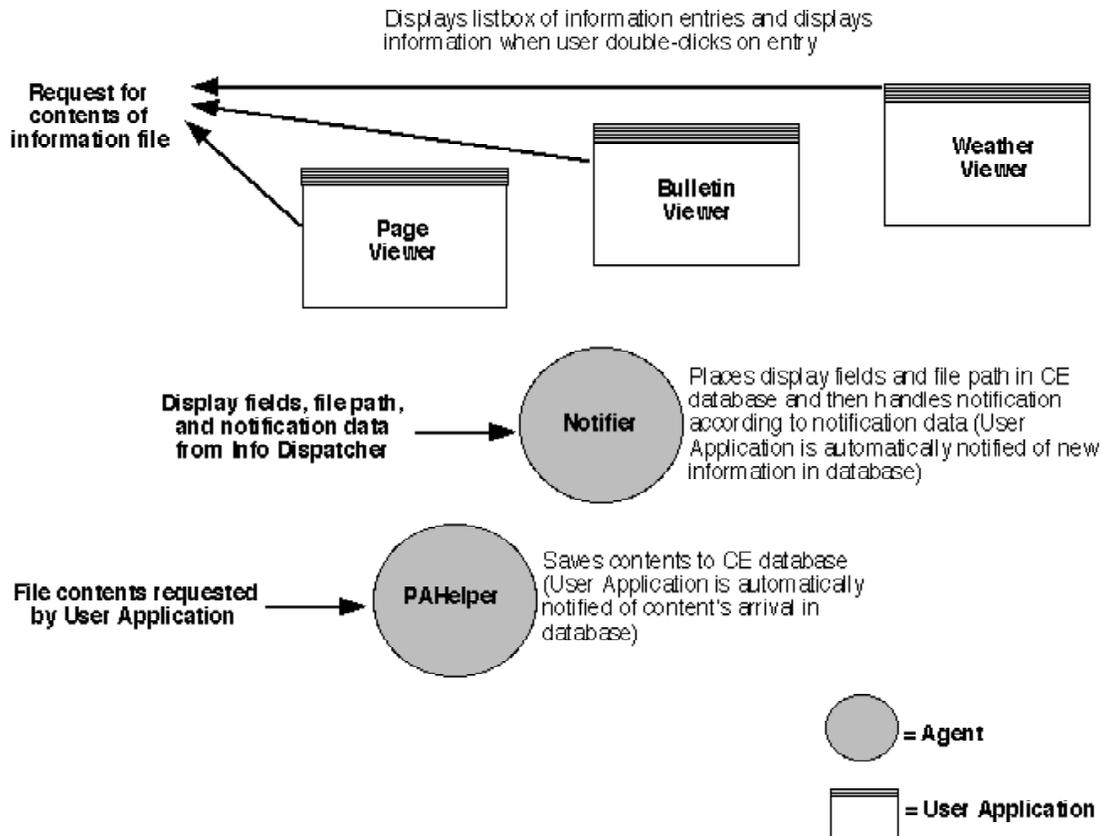= Agent

= User Application

Figure 5: The Mobile Device

The information servers implemented in this project are a weather server, a

paging server, and a bulletin server. A simple way to implement these servers would be

to write Perl scripts that periodically receive requests from the Personal Agent over a

socket, parse a specific web page to obtain needed information, and then send the

information back over the socket to the Personal Agent.

In the implementation of the project, however, the Information Gatherer obtains

information from pre-written files. After the Information Gatherer has obtained the

needed information, it determines the values for the fields used for personalization (i.e.,

for a bulletin the fields would be the source, subject, and urgency of the bulletin) and then

stores the actual information in a file. The thread then sends an event to the Information

Dispatcher to notify the dispatcher of the new information. This event contains the value

of the personalization fields and also contains the full path of the file containing the data.

When the Information Dispatcher receives an event from the Information

Gatherer, it carries out the personalization specified by the new information's associated

personalization profile. The personalization profile could delete the information if it

determines that the information will not be useful to the user. If the personalization

profile does not filter out the information, then it will produce data that describes how the

Notifier should handle notification of the new information. Any information that the

profile does not delete will be available to the user. The system, however, may or may

not explicitly notify the user of the new information. The Information Dispatcher sends

this notification data, the fields the viewing applications will display for the new

information, and a pointer to the file that contains the actual information. This pointer is

simply the file's full path on the Personal Agent's host machine.

When the Notifier receives the above data, it immediately supplies the viewing

application associated with the new information, including the information's display

fields and the pointer to the name of the information's file. This notice allows the user

application to create and display an entry for the new information. The Notifier then

analyzes the notification data sent by the Information Dispatcher. This data includes a

list of the activity categories and the list of locations under which the notification can be

posted. If both lists are empty, then the Notifier knows that a notification should never be

posted. The data also includes the message that should be displayed during the

notification. If the user need not be notified of the new information, then the Notifier is

done processing the information. If the user needs to be notified, then the Notifier must now check if the user's current context satisfies the information's context criteria. The Notifier is always aware of the user's current context because it is informed of any context changes by the Calendar application and the Location Detector. If the current context satisfies the information's context criteria, then the Notifier displays a message box containing the notification message. The Notifier is then done processing the new information. If the current context does not satisfy the information's context, then the Notifier places the notification data into a list of notifications that must still be processed. Each time the Notifier receives a context change, it runs through the list to see if the current context satisfies any of the notifications' context criteria. For each one that is satisfied, the Notifier posts the appropriate notify to the user and removes the notification data from the list.

The user applications currently integrated with the Personal Agent System are a Weather Service, a Paging Service, and a Bulletin Service. Each of these user applications has its own database and when the Notifier receives new information, it writes the display fields and file pointer of the information to a new entry in the appropriate database. The user application automatically receives notification of the new entry and, in its listbox window, the user application creates an entry for the new information. When the user double-clicks on a listbox entry, the user application obtains from the database the pointer to the information file. The application then sends this pointer and the id of the information's database entry to the HHHelper. The HHHelper receives the information file request, opens the file specified by the pointer, and sends the contents of the file and the database entry id to the PAHelper. When the PAHelper

receives this event, it saves the contents of the file into the database entry specified by the id. The user application automatically receives notification of the change to the database entry. It reads the contents of the file from the database entry and displays it in a pop-up window. When the user closes the pop-up window, the user application deletes the contents of the file from the database entry since data should be stored only with the Personal Agent.

The Calendar used in the implementation reads a pre-written schedule from a file on the hand-held. A schedule consists of the begin and end time of each activity and specifies a category for each activity. The Location Detector is a stub that sends a different IP address to the Notifier at pre-selected times.

As this implementation section shows, the Personal Agent System provides an infrastructure for using context detectors to personalize information retrieved from information servers and pushed to user applications located on a mobile device. Information servers, viewing applications, personalization profiles, and context detectors are all easily integrated into the system. Together they provide mobile users with a personalized information retrieval system that manages the vast amount of information available and provides a convenient and minimally intrusive interface to the users.

5. Personalization

The method of personalizing this information gathering system to a user is an area that can be explored in depth. Personalization involves a variety of concepts—including concepts in user-interface design, machine learning, information retrieval, information

filtering, and context awareness. We focus here on rule-based systems and learning models used in designing the Personal Agent System's learning mechanism.

5.1 Rule-Based System

We designed the Personal Agent System to support personalization profiles that are a collection of rules that establish how a specific information type should be personalized. For example, the weather service would have an associated profile for personalizing incoming weather data to the user's preferences. Having user preferences about a specific data type encoded in a module such as a personalization profile greatly increases the flexibility of the system—it allows the system to be easily changed and extended. If the user would like to personalize the retrieval of stock quotes, for example, then she would just need to add a stock profile to the collection of existing personalization profiles. Profiles could even be pre-configured to represent a certain user type. A user could select a pre-constructed profile that most closely matches her preferences and then she could either explicitly update the preferences encoded in the profile or could have them automatically updated through the learning mechanism that we describe later.

We selected a rule-based system as the system for encoding user preference's into a profile because such a system is extensible and is a representation into which an application can easily translate user preferences. This ease of translation exists because rules directly associate conditions into desired actions and also provide a straightforward way of organizing many preferences. We can, for example, simply have preferences listed in order of specificity or importance. Furthermore, we can easily update a list of rules by adding more rules or changing a specific condition in an existing rule.

Therefore, having preferences encoded as rules is well suited for handling preference changes--either directly from the user or through the learning mechanism.

The main components of a system of rules are conditions, actions, and defaults [1]. To determine how it should personalize incoming information, the system starts at the top of the list of rules and stops at the first rule whose conditions, represented as Boolean expressions, are satisfied. The conditions involve the values of the personalization fields extracted from the information being processed. For example, when processing a news bulletin, the conditions involve the value of the bulletin's source, subject, and urgency. The system stops at the first rule in which the conditions have the same value as the personalization fields of the bulletin. The condition has an associated action that describes whether the information being processed should be deleted, whether it should be converted into another format (i.e., a large picture might be reduced to the size of the mobile device's screen), and both whether and when the user should be notified of the incoming data. The rules are organized in order of importance and specificity and the last rule is a default rule that handles any incoming information that does not match any of the other rules. This structure allows the system to handle all information systematically. The structure of this rule-based system is directly analogous to the structure of programming languages such as Awk and Prolog and of the "switch" and "if-else" programming constructs.

For the Personal Agent System, one possible language for expressing personalization rules is Tcl. Tcl is an interpreted language that can run on Windows NT and that interfaces easily with C/C++. Below is sample Tcl code that represents two

rules in a personalization profile associated with pages (one-way electronic messages).

An application that presents a user interface for entering preferences generates this code.

```
if { $sender == "Sally Brown" && [ string match
"*performance*" $subject] && $urgency  < 3} {
     if { filterPage $filename} {
          pageToASCII $filename
          set notifyActiv { leisure eating}
          set notifyLocat { any}
          set notifyMsg "$sender has sent a \
          page concerning $subject and with an \
          urgency of $urgency."
     }
} elseif  { $sender == "Sally Brown"} {
     delete $filename
} …
```

To express the main points in this rule-based personalization system, we present

simplified personalization rules. The Information Dispatcher has already extracted the

sender, subject, and urgency fields from the page received and placed their values,

respectively, in the "sender", "subject", and "urgency" variables used in the code. The

Information Dispatcher has also placed the file path received from the Information

Gatherer in the "filename" variable. We place the Tcl rules, along with any Tcl

initialization code, into a function that the Information Dispatcher will call from its C++

code to personalize a new page.

For the two example rules, imagine that the user is in an orchestra organized by a

woman named Sally Brown. Imagine also that the user must read any of Sally Brown's

pages that are not of the lowest urgency and are about a performance. The user, however,

is not interested in any other page from Sally Brown. When the system receives a page

from Sally Brown with an urgency of 1 and the subject "performance tomorrow", the

conditions in the first example rule are satisfied. The `string match` function uses

glob style pattern matching to determine if the word "performance" is in the subject. The

matched rule performs the following personalization. First, it calls the filtering function `filterPage` to perform more detailed filtering of the page. For example, this function could analyze the content of the page to see if Sally provides the time and location of the performance. If she does not, then the function deletes the page because it knows or has learned that the user will not be interested in the page. When this deletion occurs, the function returns a 0 and the code is finished. If the filter function does not delete the page, it returns a 1 so that the `if` statement's condition is satisfied and further personalization occurs. The code then converts the page into ASCII text by calling the `pageToASCII` function. Next, the code forms a list of activity categories that the user's schedule context can be for the user to receive notification of the page. In this case, the user wishes to receive notifications when he is in at leisure or eating. The creation of a location list works the same way, but for this example the user does not have a certain location where she must be to receive the notification. Finally, the code forms the message that the Notifier will display when notifying the user of the page. The Tcl function returns the activity list, schedule list, and notification message to the Information Dispatcher so the Information Dispatcher can send this information to the Notifier.

In exploring the Sally Brown example further, consider what happens when the user receives a page from Sally Brown that either has the lowest urgency or does not have the word "performance" in its subject. In this case, the conditions of the first rule will not be satisfied but those of the second rule will be. The body of the second rule simply deletes the page—which is what the user wants. If we had reversed the order of the two rules, then all pages from Sally Brown would be deleted and the second rule would never be reached. The order of the rules is therefore critical. Rules with more specific

conditions should come before those with less specific conditions. Otherwise, the general

rule will always be satisfied before the more specific rule and therefore the more specific

rule will never have any effect on the personalization system.

5.2 Learning Models

From the example just presented, one begins to understand both the amount of

detail that rules must incorporate and the number of rules needed to achieve a high level

of personalization for the user. For this reason, we wanted to integrate a learning

mechanism into the Personal Agent System that would not require the user to manually

enter all personalization specifications. The user would have to perform this manual

entry in what Pannu and Sycara [6] refer to as a Static Filtering model.

The Static Filtering model is one of three filtering models described by Pannu and

Sycara in their paper about information-retrieving agents [6]. Although our project

performs the more general action of personalization rather than just filtering, the models

described by Pannu and Sycara are still useful in discussing the design of our project. In

a Static Filtering model, the initial setup costs are high and no learning mechanism is in

place to adjust the agent's actions. With this model, the user must undergo a tedious

initialization process and changing user interests are difficult to handle. Furthermore,

users often have difficulty expressing their interests through preferences and this model is

unforgiving of inaccurate preference settings [8]. In a Dynamic Learning Filtering

model, no initial setup occurs and the agent learns the user preferences based on various

feedback. Such a model, however, is "time consuming and frustrating" to users because

they are burdened with providing user feedback until "the agent [reaches] a reasonable

level of expertise" [6]. The third model that Pannu and Sycara describe is a Semi-

Dynamic Filtering model. This model has an initial setup phase followed by continuous use of user feedback to adjust the agent's actions.

For the Personal Agent System, the most suitable type of personalization model is a Semi-Dynamic model. This model has the user set basic preferences and then allows the system to refine these preferences to achieve more advanced personalization of services. Unlike with a Static Filtering model, the user does not have to enter all possible preferences or manually update any preference changes. Unlike with a Dynamic Learning Filtering model, the Personal Agent System can immediately provide personalized service to the user.

5.3 Design of the Learning Mechanism

In the setup phase for the "Semi-Dynamic Personalization model" of our system, users specify the learning sensitivity of the system, which information sources the Personal Agent should monitor, and how often the Personal Agent should poll each information source. For each information type, the user then specifies filtering, conversion, and notification preferences for information having certain personalization field values. The user can enter any number of these and can also rank these preferences. For example, a preference for pages could specify something like: "If the sender is 'George Smith', the subject includes the word 'dinner', and the urgency is '2', then delete the page." A preference that a user might rank higher than this preference could specify something like: "If the subject includes the word 'free', then convert the page into ASCII text and notify me when I am in class, eating, or at leisure." Note that this second preference does not filter on the sender or urgency field. Any page that has "free" in the subject will be sent to the user and posted to the user as long as he is not in a meeting.

Because the user has ranked the second preference higher, the system will send George Smith's page about dinner to the user if it has the word "free" in its subject but otherwise will delete the page. The application that obtains the user's preferences for each information type directly translates the preferences into the list of rules that form the personalization profile for the information type.

The learning phase for this model of our rule-based system consists of two different mechanisms. The first type involves learning if certain notifications are useful and if certain information is even wanted. This type of learning relies on feedback from a "Unwanted Information", "Intrusive Notify", and "Thanks" button that appear on every message box that notifies the user of new information. The "Unwanted Information" button means that the user finds the new information uninteresting and does not want it again. The "Intrusive Notify" button means that the user wants the new information but finds notification of the information intrusive. When the user finds the new information interesting and the notification of the information useful, then he would click the "Thanks" button. The user clicks any of the buttons to close the notification box and "Intrusive Notify" is the default. We chose the "Intrusive Notify" as the default because if the user must rely on the default, he most likely finds the notification intrusive and wishes to close it quickly. Assuming that the user does not want the information is too strong of an assumption. With only three buttons to respond with, the interface does not demand too much from the user.

Because the system always knows the user's context and therefore can record the context under which a notification is posted, it can receive clues about why a notification was not useful. Because the system can determine which rule handled a notification, it

receives clues on how to prevent further postings of intrusive notifications and further

obtaining of unwanted information. If the user consistently clicks the "Intrusive Notify"

button on a notification posted under a certain context, the system will remove the

context from the context list generated by the rule. If the user consistently marks

information with the same personalization fields and of the same type as unwanted and if

the personalization fields do not precisely match every condition in its matching rule,

then the system infers that matching rule is too general. In other words, the rule probably

matches too broad of an information category because it does not have enough conditions

or its conditions are too easily satisfied (i.e., it has a $\geq$ condition rather than an $=$

condition). The system therefore forms a new rule whose conditions exactly match the

value of the consistently unwanted information's personalization fields. The rule's action

will consist simply of deleting the data. The system places this new rule before the more

general rule.

The second type of learning mechanism involves learning what information the

user finds interesting. This mechanism adjusts the filter function called within a

personalization rule. This function deletes the information if it decides that the user

would not be interested in the information or it allows further personalization of the

information if it decides the user would be interested in the information. For weather

reports, the filtering function could be as simple a function that deletes a report if it does

not contain the word "tornado". For information types such as news bulletins, however, a

much more complicated filter function might be needed to analyze the information's

content.

Much research has been conducted on content-based information filtering. Pannu and Sycara suggest having the user provide a collection of information which they are interested in (positive examples) and a collection of information that they are not interested in (negative examples) [6]. They then suggest using either term frequency-inverse document frequency weighting (tf-idf), a technique that utilizes the occurrence properties of various terms, or a neural network to analyze the information collections [6]. A filter allows information through if the information's analyzation more closely matches that of the positive examples and does not allow information through if its analyzation more closely matches that of the negative examples. In their research, Pannu and Sycara found tf-idf to be more effective.

Winiwarter suggests the use of evolutionary adaptation to determine information of interest [8]. For each of her interest domains, the user must supply a collection of information that represents the domain. From this collection, the system creates "a ranked list of descriptors" that it will use to categorize new information into an interest domain [8]. From these collections, the system also derives the initial population of "chromosomes" used in the evolutionary algorithm. For each interest domain, the system derives an initial population from random variations of the collection of information representing the interest domain. When the user receives incoming information, the system categorizes it into a domain and then asks the user to rate the actual relevance of the information to the domain. The system uses the user's rating to determine the fitness of the chromosomes for the corresponding interest domain and then runs the evolutionary algorithm to adjust the system's model of the user.

In the GroupLens Research Project, the researchers integrated filterbots into a collaborative filtering system [7]. A collaborative filtering system is one in which information is filtered based on the opinions of other users [7]. This system would therefore apply more to news bulletins and web documents rather than information such as e-mails, pages, or weather reports. Filterbots are "automated rating robots" that evaluate new documents [7]. They help address the problem of trying to filter information that has not yet been rated by other users. In the project, filterbots that the user consistently agrees with are given a high weight when filtering information for that user.

Determining what properties of the network or of the incoming information the system should use for personalization is another aspect of designing a personalization service. Quality of Service (QoS) factors such as network bandwidth, noise, cost, and delay can be used to determine when information should be pushed to the user and perhaps what type of conversion the information should undergo. If the system keeps a history of the personalization fields of past information gathered, personalization could also occur based on the amount of change that occurs between incoming information. For example, a weather report could be filtered through only if it has a temperature difference of more than 5 degrees or a stock quote could be filtered though only if it has dropped more than 10 points.

6. Summary

This paper presented the design of a mobile-agent system that provides a mobile user with a personalized information retrieval service and described the implementation

of such a system's infrastructure. This project integrated research in information retrieval, information filtering, personalization, mobile agents, context-aware applications, and mobile computing. The Personal Agent System not only addresses the resource-limitation problems associated with mobility, it actually takes advantage of the characteristics of mobility to provide an enhanced service to the user. Because the Personal Agent System recognizes what information the user is interested in, converts information into a form the user can most easily view, presents the information only when the user can use it, and offers its services in virtually any location, the system indeed allows the user to take advantage of the vast amounts of information available in this exciting age of information.

Acknowledgements

I would like to thank Amanda Eubanks, Neha Narula, and Tiffany Wong for working with me on this project. I would also like to thank Guanling Chen for both his work on the *Pokegent* system and his help integrating the *Pokegent* system into the Personal Agent System. Finally, I would like to thank Professor David Kotz, my thesis advisor, for guiding me throughout this project and, ultimately, for making this project possible.

References

[1] Stefan Arbanowski and Sven van der Meer. Service Personalization for Unified Messaging Systems. In *Proceedings of The Fourth IEEE Symposium on Computers and Communications*, 1999.

[2] Susan T. Dumais. Combining Evidence for Effective Information Filtering. In *Proceedings of the AAAI Spring Symposium on Machine Learning and Information Retrieval*, 1996.

[3] Esteban Chavez and Rudiger Ide and Thomas Kirste. SAMoA: An experimental platform for Situation-Aware Mobile Assistance. In *Proceedings of Workshop on Interactive Applications of Mobile Computing (IMC'98)*, 1998.

[4] Robert S. Gray. Agent Tcl: Alpha Release 1.1. December, 1995. Available at http://agent.cs.dartmouth.edu/manual/doc.1.1.ps.gz.

[5] Rui José and Nigel Davies. Scalable and Flexible Location-Based Services for Ubiquitous Information Access. In *Proceedings of First International Symposium on Handheld and Ubiquitous Computing, HUC'99*, 1999.

[6] Anandeep S. Pannu and Katia Sycara. A Learning Personal Agent for Text Filtering and Notification, 1996. Available at http://www.cs.cmu.edu/~softagents/papers/kbcs96.ps.gz.

[7] B. Sarwar and J. Konstan and A. Borchers and J. Herlocker and B. Miller and J. Riedl. Using Filtering Agents to Improve Prediction Quality in the GroupLens Research Collaborative Filtering System. In *Proceedings of the ACM Conference on Computer Supported Cooperative Work (CSCW)*, 1998.

[8] Werner Winiwarter. PEA-A Personal Email Assistant with Evolutionary Adaptation.

In *International Journal of Information Technology*, 1999.