An Architecture for Supporting Personalized Agents in Appliance Interaction

Olufisayo Omojokun
omojokun@cs.unc.edu
Department of Computer Science
University of North Carolina at Chapel Hill
Chapel Hill, 27599-3175

Charles Lee Isbell, Jr.
isbell@cc.gatech.edu
College of Computing CRB 380
Georgia Institute of Technology
801 Atlantic Avenue
Atlanta GA, 30332

Prasun Dewan
dewan@cs.unc.edu
Department of Computer Science
University of North Carolina at Chapel Hill
Chapel Hill, 27599-3175

Abstract

An important component in the vision of ubiquitous computing is universal interaction: the ability to use arbitrary interactive devices, such as cell phones and palmtops, to interact with arbitrary appliances such as TVs, printers, and lights. We believe that these interactive devices can and should enable personalized agents to learn about the real-world behavior of their users by observing the appliance operations they invoke. The agents can then apply this knowledge to support useful and interesting features, including: (1) predicting appliance related tasks and automatically performing them on the behalf of users and (2) presenting appliance interfaces that reflect the situational preferences of users as inferred from their past interactions. In this paper, we motivate and present an architecture for integrating personalized agents in our universal interaction infrastructure. Specifically, we present the following: (1) reasons for supporting universal interaction and personalized agents in this domain, (2) a general architecture for universal interaction, (3) a framework for supporting personalized agents on top of this general architecture, (4) the current state of our implementation of this framework, and (5) open research issues we are currently exploring.

1. Introduction

Ubiquitous computing—sometimes referred to as pervasive computing—is a relatively new concept in computer science. Simply put, it is the notion that some form of computability can be present in all aspects of daily life. Its goal is to embed interactive computers of some sort everywhere: in furniture, walls, all forms of electronic equipment, and even the body of animals and humans. By moving the computer away from the bounds of the desktop, the hope is that we can discover more interesting and dynamic applications.

One such application is *universal interaction*, the ability to use arbitrary interactive devices—some of which we traditionally think of as computers (*e.g.* handhelds), and some of which we do not (*e.g.* cell phones)—to interact with arbitrary remote appliances such as TVs, printers, and lights. We will refer to these interactive devices as *interaction computers* when they are capable of presenting user interfaces (UIs) that can be used to control other devices. Figure 1 shows an example of universal interaction.

Figure 1. A presenter controlling a TV, VCR, and projector with a handheld.

Here, the user is using an HP 680 (Jornada) handheld to control a TV, VCR and projector. The "networked" appliances in this experiment consist of conventional appliances connected to a networked laptop that generates the infrared signals understood by the three appliances (via a special device connected to its serial port). The laptop exports remote appliance Java objects representing the devices; that is, objects on which remote computers can invoke operations to control the associated appliances. The HP680 provides a UI to interact with the appliances. In the future, TVs, VCRs, and other appliances can be expected to be networked computing devices that can support direct control by an interaction computer, hence avoiding the need for an intermediary device such as the laptop in Figure 1.

This vision raises several infrastructure-related issues. For example, how does an interaction computer discover an appliance? How is the specific UI for a specific appliance deployed on the interaction device? In this paper, we will briefly discuss these issues; however, we will focus mostly on exploring the possibilities of integrating personalized learning in this domain. In particular, we believe that interaction computers can learn about the real-world behavior of their users by observing

the appliance operations that users invoke. Furthermore, we believe that this behavior is consistent and interesting enough that the interaction computers can apply this knowledge to anticipate and support their users' interaction with their devices.

The rest of this paper is organized as follows: Section 2 presents a set of reasons for supporting universal interaction, which includes motivation for integrating personalized agents in this domain. Section 3 describes a general architecture of systems supporting universal interaction. In Section 4, we present a framework for supporting personalized learning in such architectures. We describe the current state of an implementation of this framework we are building in Section 5. Section 6 discusses several open design and machine-learning related issues, raised in this process, that we are currently exploring. We propose experiments that we will use to address these issues in Section 7. Finally, section 8 and 9 present related work and conclusions respectively.

2. Motivation

Today, it is possible to control appliances using conventional remote controls. Some of these controls can even interact with multiple devices such as TVs, VCRs, cable set-top boxes, and CD players. In fact, they are usually called "universal" remote controls. Why replace traditional remote controls with interaction computers? There are several advantages of adopting this approach:

- *Automatic binding to appliances*: Traditional universal controls must be manually bound to appliances by entering appropriate codes for the appliances. This is not a serious problem when the number of appliances is small; however, it would be a serious problem in a world with ubiquitous computing. Because an interaction computer is intelligent, it can bind itself automatically to any available appliance.
- *More universal*: An interaction computer would be a more universal control than a traditional remote control, for two reasons:
    - *Arbitrary number of appliance instances*: A traditional universal control talks to a fixed number of instances of an appliance, which is determined by the number of physical buttons and other controls on the device. An interaction computer can control an arbitrary number of appliances upon discovering them.
    - *Control of dissimilar devices*: A traditional universal control must provide physical controls for the union of the operations of the appliances it can control. This can clutter the control if the appliances share few operations. Therefore it typically controls similar appliances, that is, appliances such as TVs and VCRs that share a large number of operations. Dissimilar devices such as TVs and projectors require separate controls. A survey shows that 44% of households in USA have up to six remote controls (USA Today, November 2000). An interaction computer can serve as a single control for arbitrary kinds of appliances.
- *More remote:* Because an interaction computer can interact with a networked appliance over the Internet, it can be used to control an appliance from an arbitrary location. For instance, a family at the beach can use their cell phone to turn off their home water sprinkler if it becomes cloudy and start the home air conditioning from their car on their way home.
- *More control:* The three reasons given above are driving the research in this field. Perhaps a more intriguing motivation for universal interaction is that it is possible to create software UIs on an interaction computer that are more sophisticated than the physical UIs offered by traditional controls. We have identified the following kinds of enhancements that interaction computers can offer:
    - *View appliance output*: Unlike a conventional remote control, an interaction computer is an output device, and thus, can display application output such as car diagnostic readings and water sprinkler settings. The ability to display output on a remote control may not seem important when it can also be displayed on a device connected to the appliance, but there are at least two situations under which this feature is useful. First, the output device may be used to display other information of interest. For example, the TV may be showing an interesting program while the VCR settings are being input and displayed on the interaction computer, instead of displayed the TV. Second, and more important, the appliance data sometimes needs to be viewed in an offline mode, when the interaction computer is no longer connected to the appliance or its user is no longer within sight of the output device. For example, car diagnostic output and water sprinkler settings may be viewed in the offline mode at a location remote from the one in which the appliance is situated.
    - *Offline editing and synchronization:* Appliance data can also be edited in the offline mode, and later synchronized with the appliance. For example, a person can edit the water sprinkler settings in the offline mode and then later synchronize them with the appliance. This facility has been found to be useful in some traditional computer-based applications such as address books and, as this discussion shows, can also be useful for appliance interaction.
    - *Multi-appliance interaction:* An interaction computer can provide commands that can be invoked on multiple appliances. For instance, it can allow a user to automatically turn off all lights in a building or all appliances in a house except the refrigerator. Similarly, an interaction computer can send audio output it receives from

an appliance to a stereo system and video output to a projector (Han, Perret, and Nagshineh 2002).

- *Personalization*: One of the expectations of universal interaction is that users will own an interaction computer that they will carry with them in order to manipulate appliances as they move across ubiquitous computing environments. We believe that interaction computers can learn about the real-world behavior of their users by observing the operations they invoke and then apply this knowledge to better support the user. This can be done by deploying an agent with learning algorithms on the interaction computer in order to observe the activities of its user, either directly through various sensors, or indirectly by asking the interaction computer to provide a stream of events. We propose some possible features that could be provided by such an agent below:
    - *Preference storage:* Through a personalized agent, the interaction computer can automatically feed user-specific data to shared appliances, such as favorite channels and volume levels to a TV, PIN number to an ATM machine, credit card number to a snack machine, preferred car-seat tilt angle to a car, and files to a printer. Some of these data such as PIN and credit car numbers can be recorded during the first interaction with the appliance and replayed in later interactions.
    - *Task Automation:* Moreover, personalized agents can observe the activity patterns of their users and use this to predict future activity. Subsequently, they can automatically act on behalf of their users by invoking predictable operations. For example, a personalized agent should learn that its user tends to watch a certain television channel at a certain time daily by observing past commands invoked though an interaction computer. It could then begin to perform the task of setting the nearest television to the user to the certain channel at the appropriate time. Furthermore, it may be useful for an agent to observe behavior that is out of the ordinary or inconsistent with its user's usage pattern. For instance, a personalized agent could notify a person to set a clock alarm if that person normally sets the alarm each day and forgets one day. Also, if a homeowner that normally activates a security system each night de-activates it one night when really trying to activate it, a personalized agent should confirm the homeowner's decision before sending the operation to the security system. Based on anecdotal evidence of the authors' own appliance usage patterns, we believe that there is enough regularity that such learning can be achieved.
    - Another interesting application of personalized agents in universal interaction is in customizing appliance UIs to reflect the needs of users and the capabilities of the interaction computer. For instance:
        - *Filtering*: An appliance UI could present only the most useful commands of an appliance at any given time. This is especially useful here because interaction computers can be small and resource-limited. Many of today's Microsoft applications display only the most recently used menu items as a means to simplify their UIs. With personalized learning, however, we can be more sophisticated, tailoring the UI to what we expect the user will need at a given moment based on the statistics of the current situation and the user's past interactions when in similar situations.
        - *Pre-fetching:* A personalized agent could pre-fetch the UIs of other appliances that are commonly used with the appliance with which the user is currently interacting. For example, once a presentation speaker chooses to control the projector, a personalized agent on the handheld device could automatically fetch the UIs for the lights in the room because it learns from the speaker's history that lights are usually dimmed each time a projector is in use. Another common scenario in which this feature might be of use is in home-theatres (TV, DVD, A/V receiver, etc).

The next section presents a general architecture for universal interaction from which we design a framework for supporting personalized learning functions like those highlighted above.

3. General Architecture

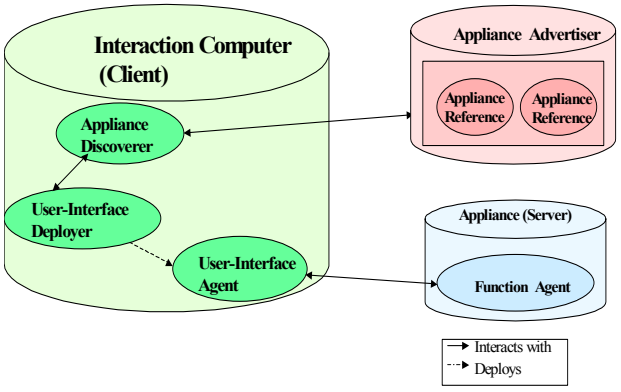

Figure 2. Universal Interaction Architecture. It consists of three major components: an appliance containing a function agent that implements its operations, an appliance advertiser that advertises references to the appliance over the network, and an interaction computer that discovers appliances and deploys a user interface for them.

Figure 2 shows the general architecture of an infrastructure supporting universal interaction. It contains the following components: function agents, UI agents, appliance advertisers, appliance references, appliance discoverers, and UI deployers. Function agents implement and perform the appliances' functions. In our example from Figure 1, the TV, projector, and VCR appliance Java objects are function agents. We use the term "agent" to refer to any active entity capable of executing code (such as a process or an object). UI agents execute on interaction computers and provide UIs to interact with appliances. They send appropriate messages to function agents to request the execution of the operations implemented by them. Appliance advertisers publish information about appliances and references to their function agents' locations (e.g. their network addresses). Appliance discoverers on interaction computers access appliance advertisers to gather appliance references. UI deployers on interaction computers, using appliance references, deploy UI agents that present the actual UIs.

Consider how this architecture would allow the presentation speaker in Figure 1 to turn on a networked projector and dim the lights in a conference room using a handheld computer. One or more appliance advertisers publish address information about the projector and lights. An appliance discoverer on the handheld interacts with an advertiser to determine references to the two function agents for controlling these appliances. They would refer to the appliance Java objects on the laptop. If the appliances were networked machines, then their function agents may reside and execute directly on them. Once the projector and light references are discovered, the UI deployer on the interaction computer (handheld) supplies compatible UI agents. User commands that are made on the UIs are delivered over the network by the UI agents as requests to the projector or light function agents.

4. Personalized Agent Architecture

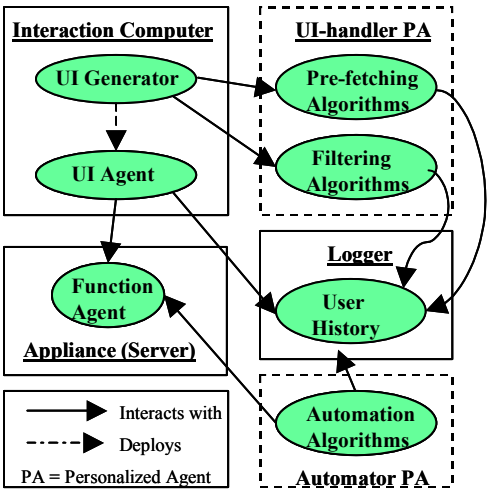

Figure 3. A framework for supporting personalized agents. It extends the generalized architecture of universal interaction (Figure 2) with two components—a UI generator and a Logger.

As part of our work, we are building personalized agents that specifically perform the task automation, filtering, and pre-fetching functions discussed in Section 2. Figure 3 illustrates our framework for integrating such functions with the base architecture presented in Section 3. This framework extends the generalized architecture of universal interaction (Figure 2) with two components: a UI generator and a Logger. The UI generator generates a UI for controlling an appliance and the Logger receives and stores all commands that a user invokes on an appliance through the appliance's UI.

Because the UI-related personalization features (filtering and pre-fetching) that we are interested in require the ability to adapt an appliance's UI on the fly, this framework requires a UI generator as the UI deployer. A single UI generator resides in an interaction computer and extracts enough information about an arbitrary appliance's functionality to generate and deploy a UI. This can be achieved if an appliance provides a public description of its functions or allows a UI generator to perform introspection on it. Personalized agents that implement UI-related features communicate with the UI generator to influence the attributes of the deployed UI. We have chosen to use a single UI generator because it provides a uniform and adaptable mechanism. Alternatives, which are discussed in (Omojokun and Dewan 2001), that download a pre-written UI agent directly from the appliance or a third-party machine are unsuitable. That approach, for example, would make it difficult to log user interactions with any arbitrary appliance.

The Logger is a general mechanism to enable any personalized agent to learn about a user. Operations invoked by a user through a generated UI are stored within the Logger, which represents a user's history of interactions with appliances. Personalized agents apply their learning algorithms on the users' histories from the Logger to achieve certain goals. Hence, the Logger must provide a database-like interface for personalized agents that facilitates searching and aggregating log data. For example, an agent that performs filtering and pre-fetching may need to retrieve recently invoked commands, or gather statistics on the commands most used after activating a particular appliance.

As noted earlier, we are specifically interested in building agents that perform task automation, filtering, and pre-fetching. The Automator PA, shown in Figure 3, performs task automation by using logger statistics to predict future operations and automatically invoking them on appliance function agents. The UI-handler PA, also shown in Figure 3, uses the Logger to handle filtering and pre-fetching. Results from the UI-handler PA's algorithms provide instructions for the UI generator to follow when generating UIs (e.g. if display space is limited, omit the sleep button from the TV UI because a search on the user's log shows that sleep is simply never used).

In the next section, we discuss the implementation we are currently building to demonstrate this framework and the two personalized agents (Automator and UI-handler) of interest.

5. Implementation

The current implementation builds upon a pre-existing UI generation system designed to use the Java object system. Based on the general architecture of universal interaction, an appliance's function agent is a Java object—an instantiation of some *class* that defines its behavior. We illustrate the generation process with an example of a lamp class and the interface that the system generates:

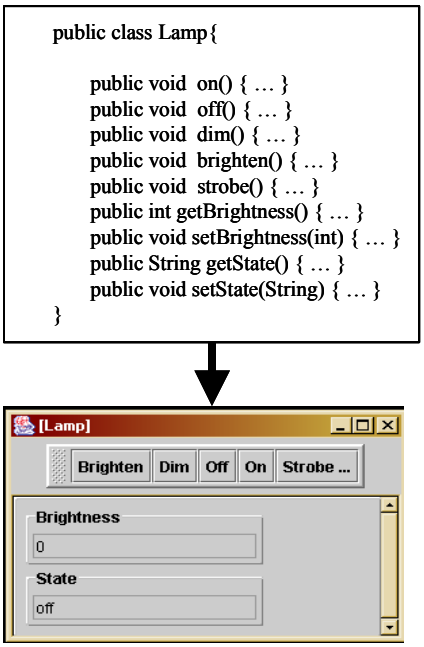

Figure 4. A lamp class outline and a picture of a sample generated UI. Methods representing commands that can be invoked on an appliance are presented as buttons on the generated UI; JavaBean properties defined through *getter* and *setter* methods are presented as form items that can be updated by both the object and the user.

As Figure 4 shows, methods representing commands that can be invoked on an appliance are presented as buttons on the generated UI. JavaBean properties defined through *getter* and *setter* methods are presented as form items that can be updated by both the object and the user. Once the user clicks a button, the appropriate appliance command is invoked, imitating today's conventional remote controls.

To support remote control of appliances over a network, we use JavaRMI (remote method invocation), which provides a remote procedure mechanism for invoking methods of Java objects on remote machines. We can support the remote control of infrared and X10 controllable appliances (such as lights, televisions, stereos, DVD players, VCRs, fans, and projectors) using networked PCs that host the appliances' function agents. Commands from the generated interfaces are sent as remote method calls to the appliance objects on the hosting PC. Invoking a method on the appliance object results in the host PC emitting the appropriate infrared or X10 signal of the appliance command. This is an unfortunate consequence of the fact that today's conventional appliances are not typically programmable and networkable (current home automation systems work similarly). Regardless, our goal is to demonstrate and experiment with personalized agents; hence, the underlying appliance control protocol is not important. Notably, in order to support remote control of appliances using this scheme, we did not have to make any modifications to the pre-existing GUI generator; we simply implemented the function agents of the appliances as Java objects.

To support the kinds of learning functions we motivate, we designed a personalized agent API, based on the components shown in the table below, and composed them with the GUI generator. The following table describes this composed API:

Implementer	Method	Return Type
UIGenerator	generateUI(ApplianceReferences)	UIAgent
Logger	log(Event)	None
UIHandlerAgent	prefetch(ApplianceReference)	ApplianceReferences
UIHandlerAgent	filter(ApplianceReference)	UI_Instructions
AutomatorAgent	start()	None

Table 1. A Personalized Agent API consisting of the programming interface defined by the components shown in Figure 3.

Before generating an interface for a given appliance, the UIGenerator calls the UIHandlerAgent's prefetch() method and passes it a reference to the appliance. The UIHandlerAgent is the UI-handler PA of Figure 3. Under our scheme, an appliance reference is the remote address of the Java object implementing the appliances functions. The prefetch() method performs pre-fetching; it returns, to the UIGenerator, a set of references to appliances whose UIs should be integrated with the generated UI. Once the UI generator receives the set of appliance references for which to generate UIs, it calls the filter() method of the UIHandlerAgent, which performs the task of the filtering. This method takes in the set of appliance references returned by prefetch() and returns a set of instructions (UI_Instructions) defining what should be presented in the resulting UI. Figure 5 illustrates a possible UI generated using filtering and pre-fetching; it merges the UIs of two appliances that are commonly used together, a TV and VCR, and presents only a subset of their commands (e.g. those most often used). For example, the TV contrast control is omitted.

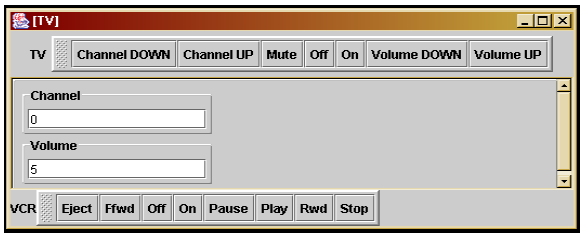

Figure 5. A possible UI generated by filtering and associating. It merges a TV and VCR UI and presents only a subset of their commands (e.g. those most often used).

Each method of an appliance object that is invoked through the UI and the Automator PA is stored as an Event in the log data structure; this is performed by having them call the Logger's log() method. An Event consists of the count and time of invocation, the kind of appliance (its class), the address of the specific appliance, the invoked method, and parameters of the method. Recall that the Logger must provide a searching interface to facilitate the operations of the external components that interact with it. As our implementation progresses, we will define the various operations needed for supporting searches and incorporate them in this API. Finally, the AutomatorAgent is the Automator PA of Figure 3; it runs independently and fires events on appliance function agents autonomously. It does not directly interact with the UIGenerator as UIHandlerAgent because it does not perform any UI-related tasks; instead, it examines and applies learning algorithms on users' logs to act on their behalf.

Personalized agents can reside locally or remotely from interaction devices; i.e., the methods that the various components of our implementation invoke among each other can be delivered over the network. An interaction device such as a cell phone may require a remote personalized agent because its local storage and processor power may be too low for logging and performing learning algorithms.

Although we have much of the pieces of the framework composed together, we must still implement the algorithms that provide the logic needed for learning. This is because we have so far focused on mechanisms rather than policies. Specifically, we have not determined what policies the filter() and prefetch() methods of the UIHandlerAgent will follow to perform their tasks, and what triggers the AutomatorAgent to automatically perform an operation. This requires experimentation and investigation of certain machine learning issues, which defines our future work. We describe these open problems in the next section.

6. Issues

Attempting to apply machine learning algorithms to our domain raises several issues and questions. This section provides discussion of some of the important ones that we are exploring.

- *Is this doable?* We must first show that appliance interaction is an appropriate domain for learning. Essentially, the question is: is there enough regularity in human appliance usage to discover patterns for learning?
- *Is there enough data?* The previous point raises an issue closely related to the problem of insufficient data. That is, in order for a typical statistical machine learning algorithm to be confident in its predictions, it requires large data sets for sampling. (In fact, proving the correctness of a learning algorithm often requires evaluation over a large or even infinite set of data.) In the case of appliance interaction, we do not expect that users can produce large amounts of data, even in their lifetimes. Suppose a user turns on a bedroom light twice a day (early in the morning and at night), this is only fourteen times a week or slightly over 700 times a year. A user should not have to switch on a light every day for two years before the system begins to automate this task. So, in order to support learning of a user's behavior within a short period such as a week, we need to explore the possibility of using fast learning algorithms (e.g. one shot learning (Yip and Sussman 1998)), to better accommodate such paucity of data.
- *Are users stable?* Given a personalized agent that is able to derive activity patterns from its user's logged behavior, it could achieve the learning-based tasks by performing frequency-based statistics on the user's log. For instance, the agent could compute which operations a user most frequently invokes at a given time of day so that it can automate them—or, which appliances does the user most often use with a certain appliance. However, our domain of appliance interaction presents the problem that users can often and sporadically change their preferences and behavior. Thus, a personal agent cannot simply perform entire history based statistics of a user's actions in deciding for an appliance: which operations it should filter from the generated UI, which appliances form associations, or what tasks it should automate. To illustrate:
    - Suppose a user often watches movies using a VCR, and a personalized agent pre-fetches the VCR's UI with the TV's. If the user later purchases a DVD player and begins to use it instead of the VCR, within some short period, the personalized agent should begin pre-fetching the DVD player UI rather than VCR's, even if the TV and VCR combination UI was appropriate for much of the past.
    - Also, suppose that a user always wakes up at 7am for work everyday with an alarm. If a week goes by in which the user sets the alarm for 8am, it may not be appropriate for a personalized agent to continue to automatically set the alarm for 7am, even though a the frequency of waking up at 7am is much greater than 8am in the user's log. On the other hand, it would be important for the agent to notice if the user appears to be returning to the 7am regime, and act appropriately.

  Together, these scenarios raise the question of how much of a user's log should an agent use in performing the statistics for its tasks.
- *Does stability trump adaptability?* One personalization function we motivate is adapting a UI to represent certain features of importance from a user's past. However, people today are accustomed to the static design of conventional remotes. Thus, we must perform some usability evaluations to show that a dynamic UI is beneficial to the user.

7. Experiments

In this section, we outline experiments that we propose to address the issues from the previous section. To determine if there is regularity in human interaction with appliances, we will provide a small set of users with interaction devices that will log their appliance interactions for some introductory period. Because we are capable of controlling common appliances found in most homes using protocols that are available today (infrared and X10), we can simply use the users' living spaces as the testing environments. Thus deploying the infrastructure in real environments is possible. After the introductory period, we can examine these preliminary user logs for evidence of regularity.

We can then begin to experiment with agents that perform pre-fetching, filtering, and task automation. The preliminary logs will be used as initial data sets. Users will then be asked to continue using their interaction devices, now integrated with personalized agents, to collect data on the performance of the learning algorithms.

Data collection will be embedded in the framework so that we can automatically collect data from remote users. For the pre-fetching algorithms, we will collect the number of times the UIHandlerAgent pre-fetches an appliance's UI who's functions are not used by the user and also the number of times it does not pre-fetch an appliance's UI who's functions are needed by the user simultaneously with another appliance (e.g. cache miss). The data collection process for the filtering algorithms will be similar to that of the pre-fetching. Essentially, we will collect the UI components (commands) that were included but not used by the user and the UI components that were not included but needed by the user. For the task automation algorithms, we will collect the number of times the AutomatorAgent fires an event on an appliance that is not desired by the user. We will require the user to explicitly notify the agent of its missteps so that we can collect such data. Finally, at the end of the data collection period, we will provide user satisfaction surveys to compare users' experiences with interaction devices and personalized agents with conventional remotes and the on-board controls of the various appliances used in the experiments.

8. Related Work

Previous work (Dewan and Solomon 1990) suggests the value of automatic UI generation in many domains. It is only recently that its role in ubiquitous appliance interaction has been demonstrated (Hodes and Katz 1998,1999; Omojokun and Dewan 2001; Ponnekanti et al., 2001). This paper extends this work by introducing the idea of integrating machine learning techniques with UI generation, which has not been considered before.

One system relevant to this work is the Remembrance Agent (Rhodes and Maes 2000), which associates files to the activity a user is actively working on. It is a continuously running program that displays information about documents that a user has seen in the past that have some relevance to text that the user is entering or reading. The Remembrance Agent shares with our work the feature of making associations with objects relating to a user's past. To the best of our knowledge, it assumes that users want associations to documents throughout their entire history or some pre-defined period. In our case, we do not always want an agent to examine at the entire history of a user; this increases difficulty in making associations because, in our case, the agent must decide how much of the history it should consider on its own.

Another relevant system is Cobot (Isbell et al 2000). It is an agent that learns the social behavior of users in a virtual chat-like environment. Using the actions that users perform on one another, it is able to predict certain attributes of a user. For instance, if a user A interacts with user B more than another user C, it should consider user A to be closer to user B than C. However, if user A later begins to interact with user C much more than user B, it must reconsider its model of A, B, C even if A has interacted B more over the long term. This example illustrates that Cobot shares some of the issues presented in Section 6.

Yet another closely related work is Coordinator (Horvitz et al, 2002), which like our work, is targeted at mobile users. Based on the history of users' actions and data about them (such as their appointments), it makes forecasts about their location, the device they are using and likelihood they will return to their office within a certain period of time. It will be useful to integrate and adapt the policies in these systems to the domain of appliance interaction.

9. Conclusions

In this paper, we motivated the use of personalized agents in universal interaction. In general, personalized agents can use the past appliance interactions of their users to predict and automate tasks for users and present appliance UIs that reflect the ongoing preferences of specific users. We presented a general architecture for universal interaction and described a framework for integrating personalized agents. Finally, we discussed the current state of our implementation of this framework, which raises several open research issues that we also discussed.

Further work is require to identify (a) whether there is enough regularity in human appliance usage to discover patterns for learning, (b) fast algorithms that we can implement in our current system to support the learning of user behavior with sparse data samples, and (c) whether users desire the features we motivate through user experiments.

10. Acknowledgements

This work was supported in part by U.S. National Science Foundation Grant Nos. IIS-997362, IIS-0228838, and ANI 0229998.

11. References

Dewan, P. and Solomon, M. 1990. An Approach to Support Automatic Generation of User-interfaces. *ACM Transactions on Programming Languages and Systems*12(4):566-609.

Han, R.; Perret, V.; Nagshineh, M. WebSplitter: A Unified XML Framework for Multi-Device Collaborative Web Browsing. In Proceedings of ACM Conference on Computer Supported Cooperative Work (CSCW), 2000.

Hodes, T. and Katz, R. 1999. A Document-based Framework for Internet Application Control. In Proceedings of the Second USENIX Symposium on Internet Technologies and Systems, 59-70. Boulder, CO: USENIX.

Hodes, T. and Katz, R. 1999. Enabling Smart Spaces: Entity Description and User-Interface Generation for a Heterogeneous Component-based System. In DARPA/NIST Smart Spaces Workshop.

Horvitz, E.; Koch, P.; Kadie, C.M.; and Jacobs, A. 2002. Coordinate: Probabilistic Forecasting of Presence and Availability. In Proceedings of the Eighteenth Conference on Uncertainty and Artificial Intelligence, 224-233. Edmonton, Canada.

Isbell, C.; Kearns, M.; Kormann, D.; Singh, S.; and Stone P. 2000. Cobot in LambdaMOO: A Social Statistics Agent.. In Proceedings of AAAI 2000.

Omojokun, O. and Dewan, P. 2001. Directions in Ubiquitous Computing: A Position Statement. In Proceedings of the Second IEEE Workshop on Internet Applications, 78-79. San Jose, California: IEEE Computer Society.

Ponnekanti, S.; Lee, B. ; Fox, A.; Hanrahan, P.; and Winograd, T. 2001. ICrafter: A Service Framework for Ubiquitous Computing Environments. In *Proceedings of the Ubiquitous Computing Conferenc*e, 56-75. Atlanta, GA.: Springer-Verlag.

"Remote Possibilities." November 2000. *USA Today*. http://www.usatoday.com/snapshot/life/lsnap180.htm

Rhodes, B. and Maes, P. 2000 Just-in-time information retrieval agents. *IBM Systems Journal special issue on the MIT Media Laboratory* 39, 3 and 4: 685-704.

Yip, K. and Sussman, G. 1998. "*Sparse Representations for Fast, One-Shot Learning*." MIT AI Lab Memo 1633.