Pers Ubiquit Comput (2004) 8: 146–153
DOI 10.1007/s00779-004-0273-z

ORIGINAL ARTICLE

Charles L. Isbell Jr · Olufisayo Omojokun
Jeffrey S. Pierce

From devices to tasks: automatic task prediction for personalized appliance control

Received: 15 February 2004 / Accepted: 9 April 2004 / Published online: 26 June 2004

Abstract One of the driving applications of ubiquitous computing is *universal appliance interaction*: the ability to use arbitrary mobile devices to interact with arbitrary appliances, such as TVs, printers, and lights. Because of limited screen real estate and the plethora of devices and commands available to the user, a central problem in achieving this vision is predicting which appliances and devices the user wishes to use next in order to make interfaces for those devices available. We believe that universal appliance interaction is best supported through the deployment of appliance user interfaces (UIs) that are *personalized* to a user's habits and information needs. In this paper, we suggest that, in a truly ubiquitous computing environment, the user will not necessarily think of devices as separate entities; therefore, rather than focus on which device the user may want to use next, we present a method for automatically discovering the user's common tasks (e.g., watching a movie, or surfing TV channels), predicting the task that the user wishes to engage in, and generating an appropriate interface that spans multiple devices. We have several results. We show that it is possible to discover and cluster collections of commands that represent tasks and to use history to predict the next task reliably. In fact, we show that moving from devices to tasks is not only a useful way of representing our core problem, but that it is, in fact, an easier problem to solve. Finally, we show that tasks can vary from user to user.

C. L. Isbell Jr (✉) · J. S. Pierce
Georgia Institute of Technology, 801 Atlantic Avenue, Atlanta, GA, USA
E-mail: isbell@cc.gatech.edu
Tel.: +1-404-3854304
E-mail: jpierce@cc.gatech.edu
Tel.: +1-404-3854239

O. Omojokun
University of North Carolina/Chapel Hill, CB3175 Sitterson Hall, Chapel Hill, NC, USA
E-mail: omojokun@cs.unc.edu
Tel.: +1-919-9606810

Keywords Ubiquitous computing · Appliance · User interaction · Machine learning

1 Introduction

Ubiquitous computing promises a future in which some form of computability is embedded in all aspects of daily life [30]. By moving the computer away from the desktop, the hope is that we can discover interesting and dynamic applications. One such application is *universal appliance interaction*: the ability to use arbitrary mobile devices—some of which we traditionally think of as computers (e.g., handhelds and wearables), and some of which we do not (e.g., cell phones)—to interact with arbitrary appliances, such as TVs, printers, and lights. Typically, each appliance has a set of *commands* that it can execute. We will refer to an appliance as an *interaction computer* when it is capable of communicating to other devices through a user interface that captures those commands. Already, there are several systems in existence that support the automatic and dynamic generation of graphical and speech interfaces for interaction computers (e.g., [21, 23]). However, these architectures provide only part of the necessary infrastructure.

Imagine walking into a room at home filled with appliances of various sorts. As you enter, each appliance broadcasts its abilities to whatever devices you happen to be carrying. Assuming that one of your devices acts as a remote control, enabling you to control other appliances; which of these appliances' interfaces should it make available to you? Following [16], we refer to this question as the *active device resolution problem*. In commercial all-in-one remote control solutions currently available, the remote control allows the user to indicate the active device by choosing from a list of available devices. In current systems, such as the Pronto, the interfaces are pre-loaded onto the remote control. However, with discovery protocols such Sun's Jini and Bluetooth, is not difficult to imagine that, in the near future, such remotes will be capable of receiving any unknown interfaces dynamically. Such a manual solution works, but is rather slow and cumbersome, especially as the number of devices grows. As we might expect, the device acting as a remote often has limited screen real estate, so scrolling and searching is often required.

In this work, we seek to address the active device resolution problem by solving what might appear, at first, to be a harder problem: the *active task resolution problem*. In this formulation, the goal of an intelligent remote control is not to determine the device that the user wants to use next, but, rather, to determine the task that the user wants to accomplish. Rather than ask whether the user wants to interact with a television, a DVD player, or the lights, such a system asks whether the user wants to use the set of commands associated with watching movies (a task that might include commands from light fixtures, the DVD player, and, perhaps, a subset of a television), with cooking, or with channel surfing.

We believe that, in general, interaction computers can learn about the real-world behavior of their users by observing and constantly adapting to how users invoke commands.

Furthermore, we believe that this behavior is consistent enough that interaction computers can apply this knowledge to personalize the user interfaces (UIs) of those appliances with which their users interact. Our main technical result is a method for discovering and modeling *tasks*, and then using those tasks as an efficient mechanism for prediction. We define tasks in a specific way, describing a representation that allows a straightforward application of clustering to discover the tasks that a user engages in. We show that using tasks is a superior mechanism for automatically building user interfaces. Finally, we show that the tasks engaged in by users are not necessarily universal, in that they vary from user to user. As such, adaptation and personalization are necessary.

In the remainder of this paper, we explore these issues in detail. In the next section, we describe other approaches to attacking this problem. In Sect. 3, we define what we mean by a task and describe an algorithm for extracting a user's common tasks. We also describe our experimental set-up and discuss results for task extraction on a collection of real-world data. In Sect. 4, we describe the prediction algorithms used for predicting the tasks discovered in Sect. 3 and present a variety of empirical comparisons showing the efficacy of our method. Finally, we discuss the results we have obtained and suggest avenues for future research.

2 Related work

There have been a variety of efforts to quantify the value of automatic UI generation across many domains [4, 5]. It is only recently that its role in universal appliance interaction has been demonstrated [10, 11, 21, 23]. This paper extends this work and the work in [22] in two ways. First, we introduce the idea of integrating a variety of machine learning techniques with UI generation. Second, we treat devices as collections of functionality that may be composed in ways that have personal meaning to each user.

Within the machine learning community, there has been a large amount of work in dealing with predicting user behavior. Of note is the early work of Hirsh and Davison [3, 9] in building an adaptive command line interface for Unix. In that effort, a variety of statistical machine learning techniques (including decision trees and Markov chains) were built into ilash, an extension of the tcsh shell, to allow for prediction of the next command to be typed by the user.

Another relevant machine learning system is Cobot [14]. Cobot 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, Cobot is able to modify its own behavior in order to maximize the long-term expected reward from each user. Of particular interest here is the choice of representation for actions. Actions are treated as distributions over the actions to follow. As we shall see in the next section, this is identical to the representation that we use for commands, and is a basis for automatically extracting tasks.

Other closely related work at the intersection of machine learning and human–computer interaction is Coordinator [13], a system targeted to 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 the likelihood that they will return to their office within a certain period of time.

Perhaps the most closely related work to what we describe here is that of [16]. In their system, they focus on active device resolution, and use a Markov model to predict the next device that the user will want to use based on the last (few) commands that the user has issued. Finally, there is the work of Mozer [18], who built a smart house that adapted to some of its users' appliance usage.

3 The active task

In Sect. 1, we defined the *active device resolution problem* as one of determining the device that the user will want to interact with next. Here, we will focus instead on the *active task resolution problem*. In this formulation, our goal is to determine the task that the user wishes to accomplish. In particular, our interest is to derive a model for prediction in a purely data-driven way. The data that we have available to us are the commands that are available for each device. As such, we will define a *task* as a collection of related commands. In this way, it is clear that a device is nothing more than one kind of task. On the other hand, with many consumer devices sporting dozens and dozens of commands, an entire device seems too large an aggregate of commands to be meaningful for our purposes. For example, when watching a movie, one will not typically use the set-up menu features of the DVD player, or even most of the menu navigation buttons. Worse, the task of watching a movie actually involves not just a DVD player, but also volume for the 7.1 surround sound system, possibly a set of lights, perhaps a retractable screen, and certainly some kind of display device. Thus, a task can, and should, span multiple devices, or at least parts of them.

So, how can we derive a meaningful subset of commands to use? One option, of course, is to allow users to define their own collections of commands and to arrange them as they wish. This is possible, but it represents a heavy burden on the user. Although the long-term benefit may be significant, the amount of short-term effort required to build such an interface is much more significant, requiring both a great deal of artistic and technical skill, and quite a bit of time.

The key here is to define what it would mean for two commands to be related, or be similar, to one another. Here, we follow [14] and suggest that two commands are similar if they predict similar behavior. In other words, a command can be viewed as implying a distribution over the commands that will follow it. Two commands are, therefore, similar if their distributions are similar. Thus, a TV channel-up button is like a TV channel-down button only insofar as both tend to predict, for example, that other channel buttons are likely to be pressed next, but the DVD play button is not.

Formally, then, we shall represent a command, $\boldsymbol{C}_i$, as a vector. Each element, C_{ij}, represents the number of times that command i is followed by command j. Thus, a command is a histogram of commands that follow it. This representation is similar in spirit to the vector space model for text [27] where a document is represented as a histogram over the words that the document contains. Following work in text retrieval and classification, we will use the standard definition of similarity between two commands as the cosine of the angle between the two vectors. By contrast, we could have used a standard Euclidean distance metric; however, because the relative proportion of counts is more important with distributions than absolute values, the cosine of the angle is typically superior.

With a formal definition of our elements of interest, and a formal notion of similarity, it becomes straightforward to derive an algorithm that will find tasks in our data. A task is nothing more than a cluster of similar commands. Therefore, the entire collection of tasks available to the user is made up of those clusters of similar commands that span the space of all commands.

Clustering is an active area of research in statistics, pattern recognition, data mining, and machine learning, and various clustering techniques are abound. In this work, we will find our tasks using the standard k-means clustering algorithm (see [17] for a review). Briefly, k-means assumes that there are k clusters that describe a data set. Beginning with k randomly selected centroids in the same space as the data, the algorithm first assigns each data point to the centroid it is closest to. After each data point is assigned, the centroid is moved so that it is at the mean of the data points assigned to it. The process is repeated until the centroids no longer change.

3.1 Experimental set-up

In order to explore the usefulness of our notion of a task, we recorded the interactions of two different users over a number of weeks with appliances from their respective entertainment centers. We wanted to avoid relying on the users' self-monitoring, so we used the set-up shown in Fig. 1. Using the Evation IRMan IR receiver connected to a laptop, we were able to automatically collect 7113 interactions for User 1 and 7152 interactions for User 2. The IRMan captures an infrared signal from a remote control and converts it to an ASCII string. Once at the laptop, it is converted it into an English representation of the command (e.g., VCR play) and stored, along with the time and date of the invocation. User 1's

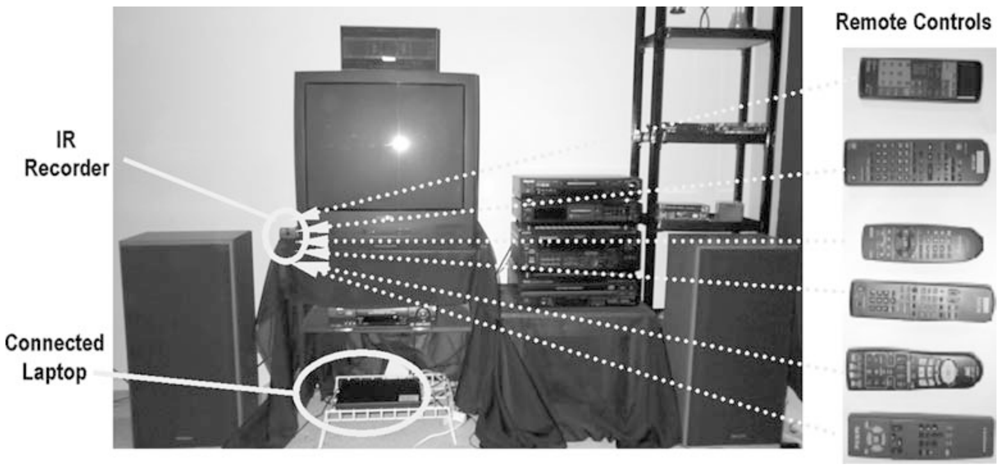

Fig. 1 Our experimental set-up. An IR receiver is placed near a set of devices and connected to a small laptop. Over several weeks, the laptop records every command invocation of every remote used by the user in the vicinity. IR signals are turned into ASCII strings and logged

entertainment center consists of a TV, VCR, A/V receiver, DVD player, MiniDisc player, and CD player. User 2's entertainment center consists of a TV, a DVD player, and a TiVo personal video recorder.

3.2 Task discovery results

Figure 2 shows a representative set of clusters discovered by *k*-means using our similarity metric. Although not all of the clusters are intuitive, the structure discovered by *k*-means is interesting.

First, many of the clusters "make sense." For example, cluster six for User 2 contains the navigation and recording keys for the TiVo personal recorder. For User 1, clusters nine and ten together contain almost all of the channel number and channel-up and channel-down keys while cluster five contains most of User 1's power and volume buttons. Cluster five is a good example of a cluster that contains commands

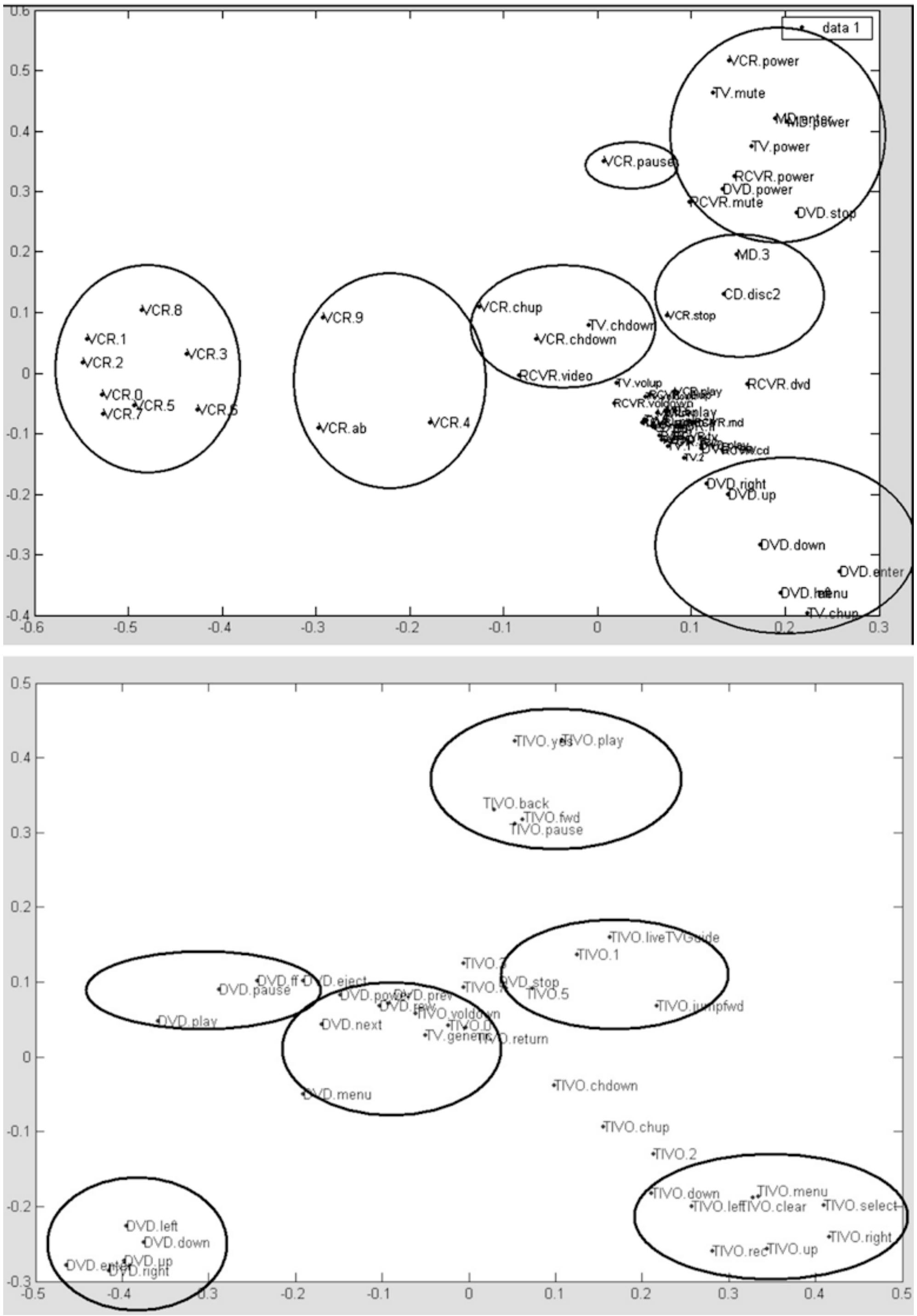

Fig. 2a, b A two-dimensional projection of related data commands with some of the clusters found by *k*-means highlighted for **a** User 1 and **b** User 2. The projection was derived using multi-dimensional scaling [28]. In this experiment, we assumed that there were 16 clusters. Notice that many of the clusters represent what we might intuitively think of as related commands from multiple appliances. In fact, the cluster has captured a useful macro: "turn on (or off) all of my devices."

Also, the sizes of the clusters vary quite a bit. The smallest clusters contain only a single command while the largest has nine. By contrast, the commands available for one of the TVs used in our study numbered 30. Similarly, the VCR had 41 distinct commands. Remembering that we expect to have little screen real estate available, this automatic way of subdividing devices seems promising. Furthermore, as we shall see below, in practice, we will be able to use these small subsets to "repair" imperfect clusters.

Finally, it is worth pointing out that some of the clusters found for User 1 and User 2 are quite different. It is possible to explain some of the variance by noting that the users own different devices; however, the differences actually run deeper. For example, both users own a DVD player, and both use the device quite often, but they use their players quite differently. Apparently, User 1 does not perform much DVD navigation, except for initial audio set-up, while User 2 spends comparatively more time exploring menu options. Quantitatively, we can construct a set of histograms for each user representing the distribution over other DVD player commands, and compare them using our similarity metric. The average cosine difference over all the DVD commands is 0.58, or about 55°.

4 Predicting the active task

Now that we have a formal notion of a task, and a method for extracting tasks from data, we turn to the problem of predicting the next active task. Our approach is to consider the user's past history of command and task usage.

4.1 Markovian models of prediction

One way of formulating our problem is that, given a sequence of events, we want to build a model that can predict the next event. In general, this is known as the *time series prediction problem*. Although we considered other possibilities, we have chosen to use a Markov process to model our time series. As pointed out in [16], it is not necessary to apply the more complex hidden Markov model because none of the state in this problem domain is hidden.

Formally, a Markov process moves through a state space. A variable, C_t, represents the value of the state at time t. For our purposes, C_t can only take on one of a finite set of values $\{1,...,M\}$. The Markov process defines the probability distribution of $\mathrm{P}(C_t|C_0,...,C_{t-1})$. The Markov assumption is that the probability of the next state depends only on a finite history of previous states. The most common form of this assumption is that the probability of the next state depends only on the value of the current state. In other words, $\mathrm{P}(C_t|C_0,...,C_{t-1}) = \mathrm{P}(C_t|C_{t-1})$. This is known as a first-order Markov model. In second-order models, $\mathrm{P}(C_t|C_0,...,C_{t-1}) = \mathrm{P}(C_t|C_{t-2},C_{t-1})$, and higher-order models are defined accordingly.

In our experiments, first-order models proved as good, or better than, higher-order Markov models, duplicating the results in [16], so each of the experiments described below assume first-order Markov models.

To complete the model, we only have to settle on a representation of the state. In our case, each state corresponds to a task, as defined by our k-means clustering. The transition probabilities are constructed dynamically from the sample traces we collected by recording the users' behavior. Specifically, we converted our histograms for each command into a probability distribution by dividing by the total number of times a command was used. Because every command indexes into a cluster, or task, it is a simple matter to compute transition probabilities for command-to-command, command-to-task, task-to-task, appliance-to-appliance, and so on.

4.2 Predicting multiple tasks

Before discussing our results, it is worth reviewing another advantage of using task-based prediction. Given a history, a Markov model induces a probability distribution over the next states. Then, normally, given a current state, a system using the Markov prediction algorithm would simply return the next state that maximizes the state transition probability from the current state.

Notice, however, that, as we have shown in Sect. 3, the sizes of the clusters found by k-means clustering vary, often containing many fewer commands than a typical appliance would. As a result, displaying a single task is not likely to use all of the available screen real estate (and if it does, then it is not clear how to present an appliance's much larger collection of commands). Thus, it seems reasonable to propose a simple extension to our prediction algorithm: rather than return the most likely next task, keep returning the most likely tasks until the number of commands they contain exceeds the available display resources. This extension may seem like an unfair advantage. In fact, the extension does represent an advantage, but also enables an apples-to-apples comparison. Modeling tasks instead of appliances makes it possible to compose commands and tasks automatically in response to the resource constraints of our remote controls.

There is another benefit to this extension as well. Some of the clusters found by our clustering split commands that are traditionally seen as being together (e.g., channel buttons). As we shall see below, by allowing the task predictor to return multiple tasks and combine them in the final interface, it is possible to bring such broken sets together.

5 Experiments and results

Using the data collected for Sect. 3, we were able to test our task prediction methods. Figures 3 and 4 contain our main results.

First, it is important to choose the best method for predicting tasks. As noted above, we could build a model that predicts the most likely tasks given a task, a model that predicts tasks given a command, or a model that predicts commands given a command. In the latter case, the last command is used to compute the next command, which is then used as an index to the task to return. To return multiple tasks, the most likely commands are collected until the union of those commands covers two or three tasks, depending upon how many tasks we allow ourselves to present. Figure 3 compares these approaches. Here, the *y*-axis is accuracy and the *x*-axis represents the size of the training set. An algorithm is considered to have provided a correct prediction so long as its prediction includes the command that the user actually used. As a baseline for comparison, we include two other obvious methods: always predicting the most common task (i.e., the most likely prior), and always predicting that the user will use the same task as they used last (i.e., the weather predictor). The task-to-task model works best and is also at least as efficient computationally as the other models.

Figure 4 compares task-to-task prediction with appliance-to-appliance prediction. The task-to-task predictor outperforms the appliance-to-appliance predictor. Also, allowing the algorithm to use more than one task further increases performance. In fact, the other task prediction methods also outperform appliance-to-appliance when they are allowed to return more than one task. Further, the task predictors need very little data to build an accurate model. The best command-to-task predictor achieves an accuracy of 90% after only 600 samples (roughly two and a half days of weekend use).

Finally, we include a sample interface that would be returned from a final system. The example, shown in Fig. 5, while not actually generated by a working device, does represent one of the most common sets of clusters that would be returned by our system using the data we collected. Notice that, even though the tasks discovered by clustering may break "traditional" clusters—for example, splitting channel buttons across tasks—in practice, an interface that combines multiple tasks will result in those traditional clusters remaining intact.

6 Discussion

In this paper, we have identified a core problem in universal appliance interaction. Specifically, we have defined the *active task resolution problem*, where the goal of an intelligent remote control is not to determine which device that the user wants to use, but, rather, which task the user wants to accomplish. Using this formulation, we have shown that it is possible to discover collections of real-world commands representing

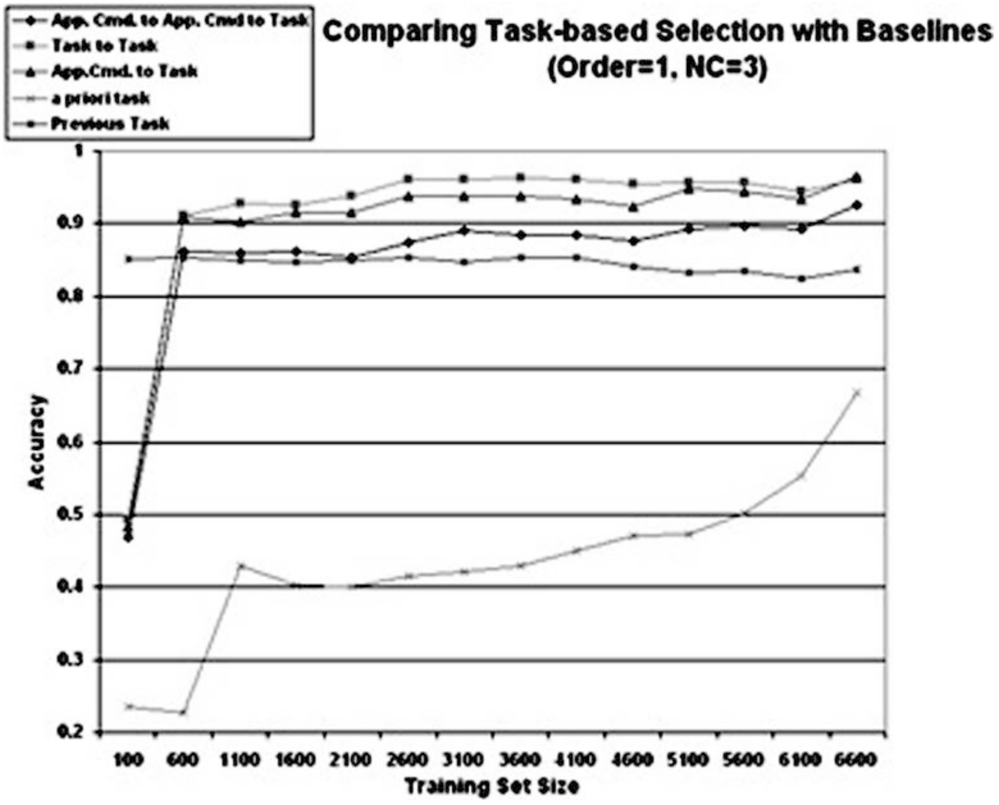

Fig. 3 A variety of possible models. In addition to predicting task-to-task directly, we could also predict command-to-task or command-to-command. Furthermore, rather than picking the best two or three tasks directly, we could take the most likely commands until we collect enough to index into two or three different tasks. Here, we show each model returning the top three tasks. The task-to-task model performs best

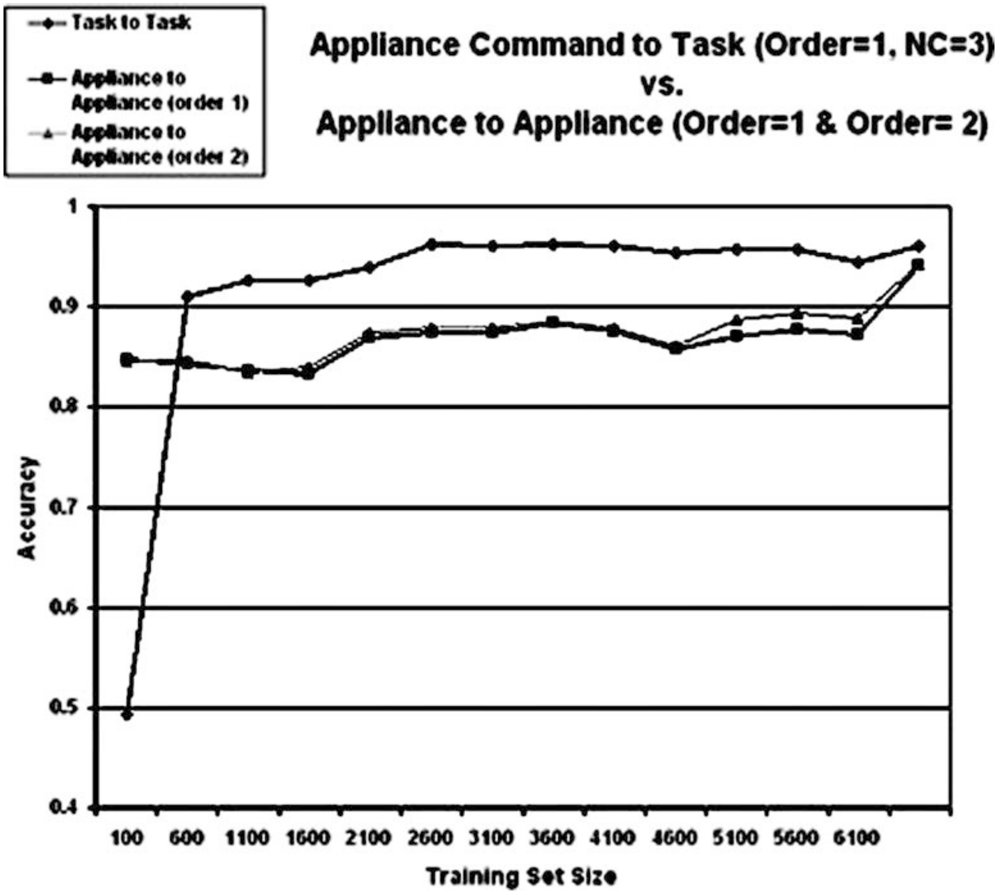

Fig. 4 Predicting tasks vs. predicting appliances. By predicting the next task instead of the next device, we are able to achieve superior performance. A correct prediction is defined as producing a set of commands that contains the next command actually used by the user. This graph demonstrates the power of using the top three most likely tasks, but also shows that only predicting the single most likely task outperforms predicting the next appliance. Also, notice that it takes very little data to achieve high performance

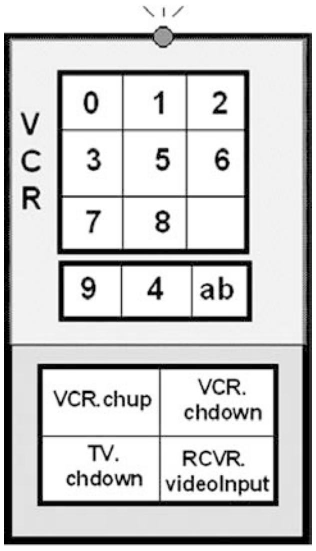

Fig. 5 A sample interface derived from our data. While our tasks may break "traditional" clusters—e.g., splitting channel buttons across tasks—in practice, an interface combining multiple tasks will result in traditional clusters remaining intact. Unfortunately, the predictor does not understand this larger relationship, so protocols must support the discovery of appliance functionality to include such information (e.g., the preferred layout relationships between various commands)

tasks, and to use history to predict the next task reliably, with an accuracy higher than previous attempts. We have also shown that tasks vary across users, so personalization is necessary.

At present, we are implementing a system for supporting personalized universal appliance interaction of the kind described here. The motivation and architectural design of our system is described in detail in [22]. Briefly, the infrastructure consists of two major components: a set of programs for predicting user behavior and a UI generator that generates the appropriate UI containing the controls needed for the predicted task(s). Our immediate plans include implementing the system described here on a handheld device and conducting several user studies to quantify the usefulness of our approach in the field.

Acknowledgements This research was funded in part by Microsoft and NSF grants ANI 0229998, EIA 03-03590, and IIS 0312328.

References

1. Brumitt B, Meyers B, Krumm J, Kern A, Shafer S (2000) Easy living: technologies for intelligent environments. In: Proceedings of the 2nd international symposium on handheld and ubiquitous computing (huc2k), Bristol, UK, September 2000

2. Chan J, Zhou S, Seneviratne A (1998) A QoS adaptive mobility prediction scheme for wireless networks. In: Proceedings of the IEEE global telecommunications conference (GlobeCom'98), Sydney, Australia, November 1998
3. Davison B, Hirsh H (1997) Toward an adaptive command line interface. In: Proceedings of the 7th international conference on human–computer Interaction (HCI'97), San Francisco, California, August 1997. Elsevier, Amsterdam
4. Desai N, Kaowthumrong K, Lebsack J, Shah N, Han R (2002) Automated selection of remote control user interfaces in pervasive smart spaces. In: Proceedings of the HCIC winter workshop 2002, Fraser, Colorado, January/February 2002
5. Dewan P, Solomon M (1990) An approach to support automatic generation of user-interfaces. ACM Trans Programming Lang Sys 12(4):566–609
6. Fox C (2001) Genarians: the pioneers of pervasive computing aren't getting any younger. Wired, November 2001, pp 187–193
7. Guttman E, Perkins C, Veizades J, Day M (1999) Service location protocol, version 2. RFC 2608, Internet Engineering Task Force, June 1999
8. Han R, Perret V, Naghshineh M (2000) WebSplitter: a unified XML framework for multi-device collaborative Web browsing. In: Proceedings of the ACM conference on computer supported cooperative work (CSCW 2000), Philadelphia, Pennsylvania, December 2000, pp 221–230
9. Hirsh H, Davison B (1997) An adaptive UNIX command-line assistant. In: Proceedings of the 1st international conference on autonomous agents (AA'97), Marina del Rey, California, February 1997
10. Hodes T, Katz R (1999) A document-based framework for Internet application control. In: Proceedings of the 2nd USENIX symposium on Internet technologies and systems (USITS), Boulder, Colorado, October 1999, pp 59–70
11. Hodes T, Katz R (1999) Enabling smart spaces: entity description and user-Interface generation for a heterogeneous component-based system. In: Proceedings of the DARPA/NIST smart spaces workshop, July 1999
12. Hodes T, Newman M, McCanne S, Katz R, Landay J (1998) Shared remote control of a video conferencing application: motivation, design, and implementation. Proceedings of the SPIE multimedia computing and networking conference (MMCN), San Jose, California, January 1999. Proc SPIE 3654:17–28
13. Horvitz E, Koch P, Kadie C, Jacobs A (2002) Coordinate: probabilistic forecasting of presence and availability. In: Proceedings of the 18th conference on uncertainty and artificial intelligence (UAI 2002), Edmonton, Canada, August 2002, pp 224–233
14. Isbell C, Shelton C, Kearns M, Singh S, Stone P (2001) A social reinforcement learning agent. In: Proceedings of the 5th international conference on autonomous agents (AA 2001), Montreal, Canada, May/June 2001
15. Johanson B, Ponnekanti SR, Sengupta C, Fox A (2001) Multibrowsing: moving Web content across multiple displays. In: Technical note in UBICOMP 2001, Atlanta, Georgia, September/October 2001
16. Kaowthumrong K, Lebsack J, Han R (2002) Automated selection of the active device in interactive multi-device smart spaces. In: the spontaneity workshop at UBICOMP 2002, Göteborg, Sweden, September/October 2002
17. Mitchell T (1997) Machine learning. McGraw-Hill, New York, ISBN 0070428077
18. Mozer M (1998) The neural network house: an environment that adapts to its inhabitants. In: Proceedings of the AAAI spring symposium on intelligent environments, Stanford University, California, 23–25 March 1998. Technical report SS-98-02, AAAI Press, California
19. Myers B, Stiel H, Gargiulo R (1998) Collaboration using multiple PDAs connected to a PC. In: Proceedings of the ACM conference on computer-supported cooperative work (CSCW'98), Seattle, Washington, 14–18 November 1998, pp 285–294. See also http://www.cs.cmu.edu/~pebbles/
20. Myers B (2000) Using multiple devices simultaneously for display and control. IEEE Pers Commun 7(5):62–65
21. Nichols J, Myers BA, Higgins M, Hughes J, Harris TK, Rosenfeld R, Pignol M (2002) Generating remote control interfaces for complex appliances. In: Proceedings of the ACM symposium on user interface software and technology (UIST'02), Paris, France, 27–30 October 2002, pp 161–170
22. Omojokun O, Isbell C (2003) Supporting personalized agents in universal appliance interaction. In: Proceedings of the 41st ACM southeast conference, Savannah, Georgia, March 2003
23. Omojokun O, Isbell C, Dewan P (2001) An architecture for supporting personalized agents in appliance interaction. In: Technical report of the AAAI fall symposium on personalized agents, North Falmouth, Massachusetts, November 2001. AAAI Press, California, pp 40–47
24. Priyantha N, Miu A, Balakrishnan H, Teller S (2001) The cricket compass for context-aware applications. In: Proceedings of the ACM annual international conference on mobile computing and networking (MobiCom 2001), Rome, Italy, July 2001
25. Rekimoto J (1998) A multiple device approach for supporting whiteboard-based interactions. In: Proceedings of the ACM SIGCHI conference on human factors in computing systems, Los Angeles, California, pp 344–351
26. Robertson S, Wharton C, Ashworth C, Franzke M (1996) Dual device user interface design: PDAs and interactive television. In: Proceedings of the ACM SIGCHI conference on human factors in computing systems, Vancouver, Canada, April 1996, pp 79–86
27. Salton G (ed) (1971) The SMART retrieval system: experiments in automatic document processing. Prentice-Hall, Englewood Cliffs, New Jersey
28. Schiffman S, Reynolds M, Young F (1981) Introduction to multidimensional scaling: theory, methods and applications. Academic Press, New York
29. Stanfill C, Waltz D (1986) Toward memory-based reasoning. Commun ACM 29(12):1213–1228
30. Weiser M (1993) Some computer science problems in ubiquitous computing. Commun ACM 36(7):74–83