Learning Automation Policies for Pervasive Computing Environments

Brian D. Ziebart[1] Dan Roth[2] Roy H. Campbell[2] Anind K. Dey[1]

[1]*School of Computer Science*
Carnegie Mellon University
{*bziebart, anind*}*@cs.cmu.edu*

[2]*Department of Computer Science*
University of Illinois at Urbana-Champaign
{*danr, campbell*}*@cs.uiuc.edu*

Abstract

If current trends in cellular phone technology, personal digital assistants, and wireless networking are indicative of the future, we can expect our environments to contain an abundance of networked computational devices and resources. We envision these devices acting in an orchestrated manner to meet users' needs, pushing the level of interaction away from particular devices and towards interactions with the environment as a whole. Computation will be based not only on input explicitly provided by the user, but also on contextual information passively collected by networked sensing devices. Configuring the desired responses to different situations will need to be easy for users. However, we anticipate that the triggering situations for many desired automation policies will be complex, unforeseen functions of low-level contextual information. This is problematic since users, though easily able to perceive triggering situations, will not be able to define them as functions of the devices' available contextual information, even when such a function (or a close approximation) does exist.

In this paper, we present an alternative approach for specifying the automation rules of a pervasive computing environment using machine learning techniques. Using this approach, users generate training data for an automation policy through demonstration, and, after training is completed, a learned function is employed for future automation. This approach enables users to automate the environment based on changes in the environment that are complex, unforeseen combinations of contextual information. We developed our learning service within Gaia, our pervasive computing system, and deployed it within our prototype pervasive computing environment. Using the system, we were able to have users demonstrate how sound and lighting controls should adjust to different applications used within the environment, the users present, and the locations of those users and then automate those demonstrated preferences.

1. Introduction

Pervasive computing is characterized by an abundance of networked computational devices embedded within our everyday environments (e.g., homes, workplaces, public spaces). Underlying infrastructures provide coordination among these devices, expanding the scope of user interaction from a restrictive coupling with specific devices to a broad interaction with the environment as a whole. As envisioned by Mark Weiser [19], this expanded interaction will allow computational devices to fade into the background of our lives, serving as a natural extension of our surroundings rather than a center of focus.

Unlike traditional electronic appliances and computer applications, which often operate isolated from and oblivious of what is occurring in the environment around them, pervasive computing applications are context-aware; they have as potential input the entire observable state of the environment [2]. This includes a partial knowledge of the physical environment obtained from sensing devices (e.g., location systems, microphones, video cameras) and a nearly complete knowledge of the computational environment (e.g., networked device state, application state, service state). Thus, the pervasive computing system is capable of basing computational decisions on users' interactions with the environment as a whole [18].

With hundreds or thousands of devices and applications within the environment, each with many different states and configuration options, automation is a necessity. Without it, users will either be overburdened in configuring the environment and updating it in response to changes, or they will leave much of the environment's functionality unchanged and only a small fraction of the environment's potential utility will be realized.

If a user's preferred action can be specified as a function of the observable state of the environment, it is possible to automate that action. For example, a user might prefer to display her email on a large room display when she is the only person present in the room, but switch to her more pri-

vate PDA otherwise. If these preferences can be easily expressed a priori by the user, tools can be used by the user to specify desired automation policies. However, the growing abundance of context sensing devices in the environment, and their disappearance into the background will make it increasingly difficult for users to understand how the context changes they observe are perceived by the environment's computing system. It's even more difficult for users to anticipate the contextual changes for an activity a priori. Experts might be capable of accomplishing this, but then users would not be in control of their own environments.

Appropriately employed machine learning techniques avoid many of the limitations of having users manually specify automation policies using tools, while still enabling end-users to automate, and enjoy the increased utility of their environments. Instead of expressing the rules for a particular action, users provide examples under varying contextual situations of when that action should be used. Users need not know anything about how the devices within the environment are sensing contextual information, nor how their perception of the environment maps to the perception of the environment's computer system. Machine learning algorithms use the user-provided examples to generate a function of the contextual information for the action that generalizes to future unseen situations. These algorithms are better suited for dealing with the wealth of low-level contextual information and determining how in combination it is indicative of the higher-level contextual information that the user perceives as a cause for an action. If the generated function errs when the machine learning techniques are employed in an online setting, a new example can be used to retrain a new function after the user corrects the error.

1.1. Our Contribution

In this paper, we present a machine learning service that observes the manual usage of the environment under varying situations in order to discover and automatically invoke user preferences in the future. Both the computational environment and the limited view of the physical environment instrumented with sensors serve as sources of contextual information on which to base training. When an adequate training set is provided by the user, machine learning algorithms generate functions to automate the environment that reflect the behavior preferences demonstrated by users. From a user's perspective, this service enables individualized automation without requiring explicit definition of desired behaviors. Instead, the user can demonstrate the concept adequately enough for the learning algorithm to correctly approximate it.

We concretely motivate the use of machine learning for creating automation policies in a pervasive computing environment with the following scenario. At varying times, an instrumented environment is used for giving small lectures, holding meetings, and as a workplace for multiple people. Depending on which activity is occurring, the environment needs to be configured differently to meet the needs of the occupants. The lighting of the environment is one aspect that needs to be configured. During presentations the lights around display devices must be dimmed to reduce glare and during meetings lights over a conference table or workplace must be more bright. As another example, when used as a workplace with multiple occupants, control of the volume of music being played is particularly important, as each occupant is willing to work with varying amounts of noise and willing to listen to the musical preferences of a subset of the other occupants. Additionally, even when only one person is in the environment the music level must change based on where that person is, lest he be unable to hear when far away from the speakers within the environment and deafened when nearby.

It is difficult for the users of the environment to define rules that distinguish between all the different activities that require different environmental configurations. Some of the same people are present for all of these activities, and only a fraction of the occupants choose to register themselves with the voluntary location system, making it impossible to associate any activity with the presence of one particular person. Similarly, the number of possible permutations makes it difficult and time-consuming to define at what volume music should be played as a function of which users are registered in the environment, where in the environment they are located, and who has chosen the music.

Instead of trying to write functions of the contextual information to control the lights and volume of the environment, the occupants use the environment as they would with no automation, changing the lights and music volume settings manually as desired. After a few days of this, the training examples they have generated are used to train functions that automate the environment in close approximation to what was demonstrated. Though the functions are complex, it turns out that characteristics of network usage vary greatly depending on how the environment is being used and the level of network traffic over a five minute window is good at distinguishing the environment as a workplace, something that the environment's occupants had not considered. There are other surprises as well. While Joe generally doesn't like any music while he is working (a well-known fact to his co-workers), right after lunch he doesn't mind at all. The environment is automated with these newly learned functions, and users manually override the automated actions with their own preferences when the function doesn't automate the environment appropriately, adding new training examples to the training set to be used for future automation. Over time different aspects of the environment

are automated with little need for future correction, allowing users to spend their time configuring other aspects of the environment.

The remainder of this paper is organized as follows. In Section 2, we describe Gaia, our current pervasive computing infrastructure. In Section 3, we present our new learning service, which we integrated into Gaia. Part of the integration into Gaia includes the creation of mechanisms for delivering contextual information to the Learning Service and mechanisms for querying the Learning Service and making use of its results. This Learning Service handles the representation of the environment and the conversion of that representation into features for learning. In Section 4, we revisit our motivating scenario and validate our Learning Service, using it to learn lighting and volume preferences based on environmental context. We conclude this paper with a discussion of the related work and our future plans for the Learning Service.

2. Our Pervasive Computing Infrastructure

The pervasive computing infrastructure, Gaia, that supports our prototype computing environment is comprised of a software layer of services that support distributed applications, and a layer of heterogeneous devices that enable user interaction. Gaia is a middleware-based meta-operating system. It runs on top of the existing operating systems (e.g., Windows, Linux) of devices within the environment to support coordination. The underlying middleware enables communication and coordination between different types of devices and programming languages. Concepts from traditional computing have been extended over the distributed host devices within the environment to transform many independent devices into one orchestrated system that, when combined with sensors and actuators, becomes a representation of the environment as a whole. Applications are built as components using an application framework and, making use of the available Gaia services and the underlying middleware, are distributed across multiple devices.

2.1. Gaia Services

Services within Gaia are distributed extensions of traditional operating system concepts. For example, the Gaia File Service [5] provides separated remote components with a unified view of data stored within the environment. The service also automatically handles the file type conversions between different systems. This service infrastructure enables the creation of applications spread over many devices that depend on the functionality these services provide.

2.2. Gaia Application Framework

Applications within our pervasive computing environment are comprised of a number of distributed components that interact with each other using event channels and remote method invocations. Applications are developed using a pervasive computing extension [16] of the Model-View-Controller (MVC) framework [10]. The extended framework specifies the roles of different components in the assembled application, and provides built-in support for application mobility, dynamic composition, and context sensitivity.

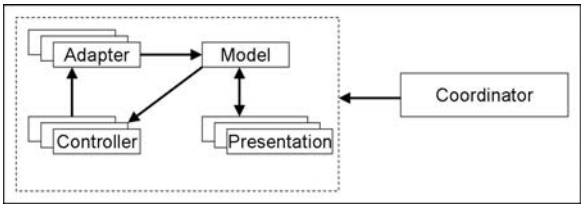

Figure 1. Application framework functional diagram

Gaia application components associate with each other functionally as shown in Figure 1. An application consists of the following components, distributed across devices within the pervasive computing environment: a model, presentations, controllers, adapters, and a coordinator. The model, presentation, and controller correspond with the model, view, and controller of the MVC application framework. The application components are distributed across the devices within the pervasive computing environment.

The coordinator manages the composition of the application and provides a means to dynamically change the application's configuration by adding or removing components as needed within the environment. The model contains the application logic and maintains the state of the application. It provides synchronization with the presentation and controller components through event channels and an interface for remote procedure calls. Presentation components serve as output mechanisms for the application. Some have visual representations, while others have non-visual effects (e.g., audio output, light and temperature controls). The presentation may respond to the model's state changes depending on the events it receives from the model, optionally invoking methods on the model to receive more information before updating its output. Controller components allow users to interact with the application logic of the model by invoking methods on the model that change the application's state. Additionally, each controller has an associated adapter that maps method calls from the controller to the methods of the model, allowing controller reuse with different models.

Each application has one coordinator, one model, any number of presentations, and any number of controllers - each with its own adapter.

2.3. Inter-application Coordination and Gaia Bridges

Applications and services within a pervasive computing environment can interact with each other in a decoupled manner using specialized components called bridges [17]. A bridge receives events from the source application or service's event channel, processes any associated data from the event, and invokes a method on the target application or service. Bridges are implemented in Luaorb [15], an interpreted language that provides bindings between Lua [6] and CORBA, COM, and Java.

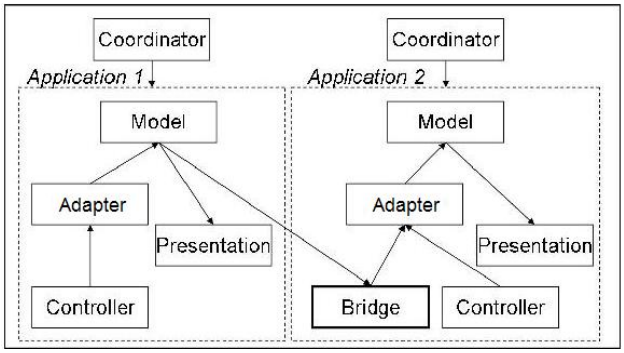

Figure 2. Application bridge between two applications

Figure 2 shows the deployment of a bridge connecting two applications. The bridge registers as a presentation of the source application (Application 1) and a controller of the destination application (Application 2). The bridge receives event notifications from the source application, possibly invokes methods on the source model to obtain additional information, and then invokes methods on the target application. As a result, the state of the target application is affected by changes in the source application's state.

3. The Gaia Learning Service

The Learning Service is integrated into Gaia, collecting information about the state of the pervasive computing environment from applications and services, learning automation policies from this contextual information, and then enabling agents to use those learned policies to automate the applications and services within environment. The mechanisms that connect the Learning Service with the applications and services of Gaia are adaptation of Gaia Bridges.

3.1. The Learning Problem

The Gaia Learning Service provides general support to learn how each contextual attribute of the environment depends on all the other contextual attributes. Each of these attributes is directly mapped into a feature to accomplish this. This feature extraction is much simpler than many learning problems, where features are functions of the raw data and experts are often required to formulate appropriate functions. However, sensors within the environment may fail and users' actions may be reported latently, so our learning algorithms must be tolerant of these sources of noise.

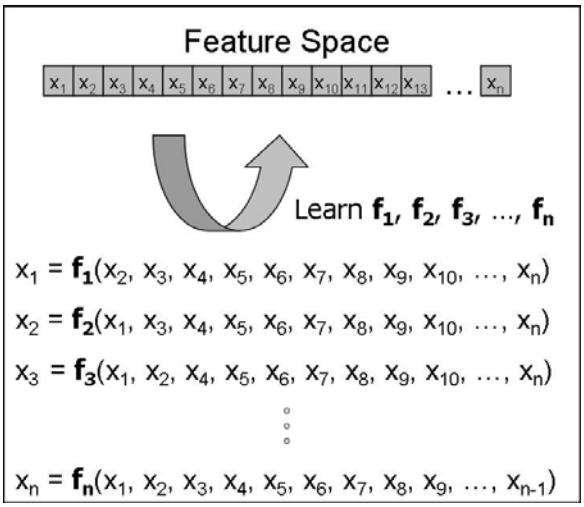

Figure 3. Learning every attribute as a function of every other attribute

Figure 3 shows a conceptual image of the learning problem. Each attribute of the environment is learned as a function of the other attributes in the environment. It isn't known beforehand which of these attributes will be of interest for evaluation, so every relationship is learned even though many may not be used.

3.2. State Representation

The state of the pervasive computing environment is provided to the learning system as sets of variable and value pairs having names that can be semantically meaningful to the programmer. The variable contains hierarchical path information to enable referencing to entities. As provided to the learning system, the variable and value pair might be: *(applications/mp3player/status, playing)*. The learning system stores these variable and value pairs in a tree structure and creates mappings between value nodes in the tree structure and unique integers that are used as features for training examples.

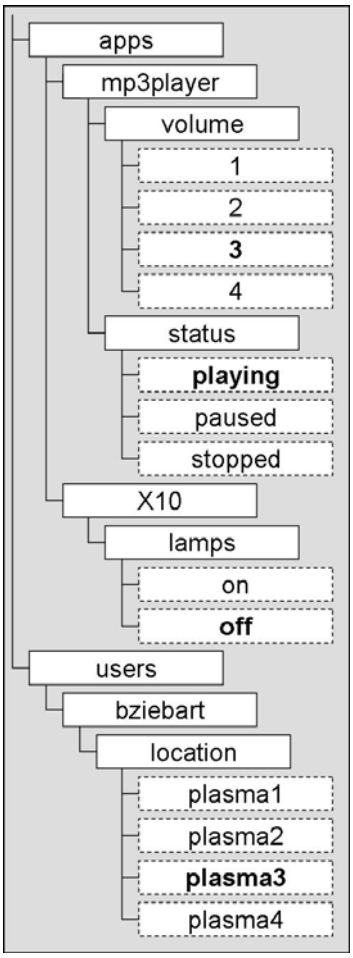

Figure 4. A sample tree representation of the environment

Figure 4 shows a possible representation of the environment in the tree structure. In this figure, variable path nodes have solid outlines, value nodes have dashed outlines, and active attributes are indicated by bold font. The Learning Service's interface provides methods to enable and disable the attributes.

3.2.1 Entities and Entity Referencing

The state representation supports the notion of entities and entity references in order to learn more generalized relationships. An entity is a set of features organized as a subtree in the tree representation of the environment state (e.g., *bziebart* in Figure 4). An entity reference is simply a path to a subtree entity (e.g., *users/bziebart*).

This reference system allows the state representation to contain abstractions of the characteristics of the environment. These abstractions enable the environment to respond to unseen entities if those entities possess characteristics of previously learned entities. For example, if Bob teaches the environment to play his music more loudly when he is at one side of the room that is farther away from the room's speakers, and softer when he is closer to the speakers, in addition to learning a correlation between Bob's location and the MP3 Player's volume, a correlation between the MP3 Player's owner's location and the MP3 Player's volume is also learned. If Jane, who has never used the MP3 Player before, begins to play music in the automated environment, the music will adjust based on her location, as Bob, and perhaps other users, have taught it.

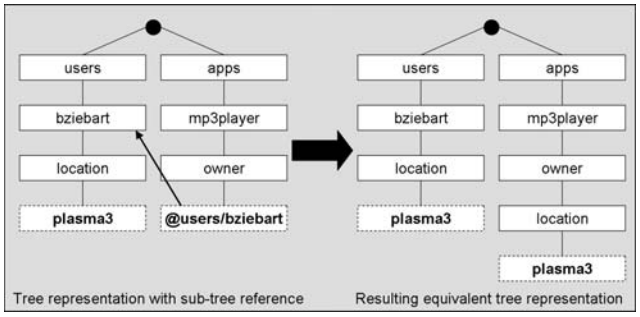

Figure 5. Entity referencing in the tree representation

Figure 5 shows the changes to the tree representation after entity reference *(apps/mp3player/owner, @users/bziebart)* is provided to the Learning Service. The reference is a pointer to the *users/bziebart* sub-tree. Training examples are generated by expanding the referenced subtree under the referrer's tree node. The resulting equivalent tree representation in Figure 5 shows this expansion for *@users/bziebart*. New unique feature identification integers are generated for these mapped values rather than using the original sub-tree identifications since the full path to this value is different in this case than in the original.

3.3. Information Providers

Data is provided to the Learning Service by components called information providers. We envision, when fully deployed within the pervasive computing environment, hundreds of information providers will supply thousands of pieces of contextual information to the Gaia Learning Service. Developers of applications and services provide the necessary specifications to generate information provides and agents, which are described later.

Figure 6 shows the three types of information providers. Information Provider 1 is essentially half of a Gaia bridge that listens to notification events from an application's model, parses these events, and delivers important changes

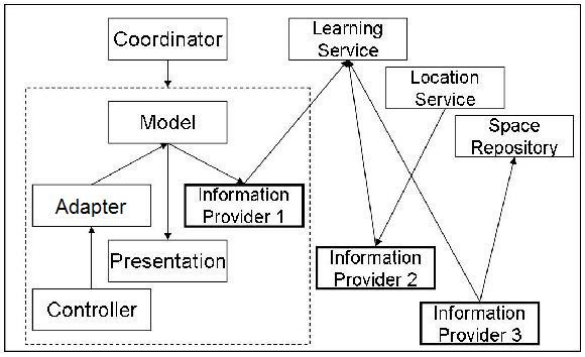

Figure 6. Three types of information providers

to the Learning Service. For example, in an MP3 Player, the information provider might deliver changes in volume, status (i.e. playing, stopped), song being played, etc. Information Provider 2 registers for callbacks from the Location Service. Whenever a registered event occurs, Information Provider 2 receives notifications and then relays the events to the Learning Service. For example, in Figure 6, the information provider may register to be notified when a particular user enters a region of the physical environment, as detected by a positioning system. Information Provider 3 periodically polls the Space Repository, which contains information about the users, services, and applications within the environment. The information provider delivers characteristics to the Learning Service that reflect any changes in the state of the Space Repository, which contains information about the applications, devices, and users within the environment.

3.4. The SNoW Machine Learning System

The Gaia Learning Service learns preferences in a batch-based manner, which consists of a training phase and a separate automation phase. During the training phase, examples are generated periodically from the representation's active characteristics. Each active characteristic is a feature in the learning system. To transition into the automation phase, the Sparse Network of Winnows (SNoW) Machine Learning System [1] is used to compute for each feature an approximate function of all other features.

SNoW is an efficiently implemented classifier architecture with support for the Perceptron, Winnow, and Naive Bayes learning algorithms. Unlike most other learning problems, where a set of labels is learned as a function of a different set of features, our learning problem learns for all features. SNoW is well-suited for quickly computing approximations for each feature.

3.5. Adaptive Agents and Applications

The learned behavior preferences can be used by querying the Gaia Learning Service and, depending on the result, invoking changes to the environment. The invoker can either be an independent agent component, or an application. Agent components are generated based on specifications provided by application developers that describe the mapping between contextual variables and invokable methods of the application. We only used a handful of agents for the experiments in this paper and they were started automatically as part of the conclusion of the learning phase, though in future pervasive computing environments we expect a user will need to have an interface to select the desired agent from a list of all available agents.

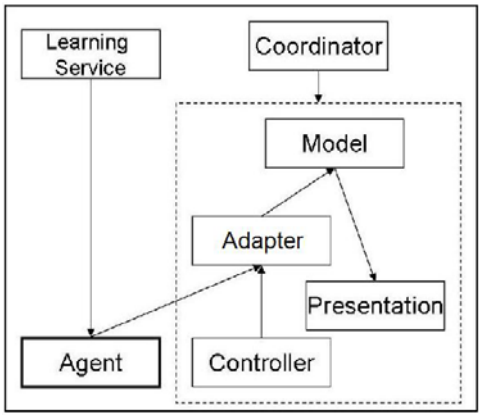

Figure 7. Agent attached to an application

Figure 7 shows an agent that uses the results of the learning service to change the state of an application. There are two types of queries that the Learning Service supports. In both types, the SNoW architecture is provided with the current features generated from the environment representation. The first option is to query for the confidence of a particular variable-value pair. The Learning Service provides a normalized activation level between 0 and 1 for the corresponding feature to be active or inactive given the provided environment state. The agent can then use this value and, if it is above or below some threshold, change the environment characteristic associated with that variable. The second option is to provide a variable to the Learning Service, and the best value choice for that variable under the current environment state will be returned. The agent can then invoke changes in the environment to reflect this best option.

4. Experimental Results

The Gaia Learning Service was successfully deployed within our prototype pervasive computing environment. We used it to learn several preferred behaviors under different domains of features. We present two of the experiments we conducted that relate to our motivating scenario.

4.1. Experiment 1: Space Repository

In the first experiment we utilized the Learning Service to automate the volume of an MP3 Player application based on the types of application components currently being used and the users present within the environment. Figure 8

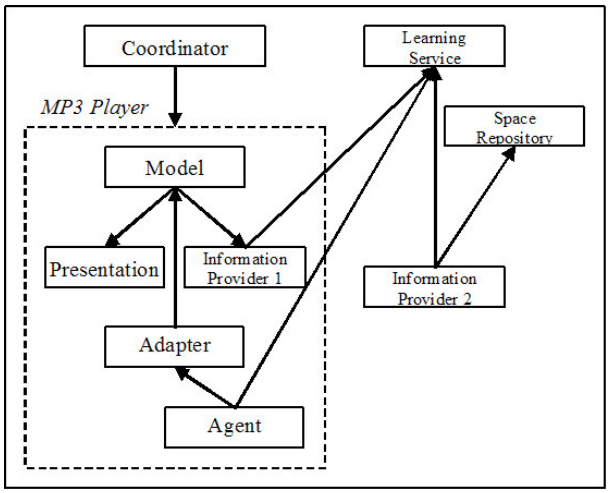

Figure 8. Experiment 1 configuration

shows the configuration for this experiment. Information providers deliver the names of application components that exist in the environment, and the volume of the MP3 Player (discretized into four different values: very soft, soft, loud, and very loud) to the Learning Service. The information provider connected with the Space Repository queries the Space Repository every three seconds. Then, during the automation phase the agent queries the variable for the MP3 Player's volume and sets the volume based on the result every three seconds.

Using this setup we taught the Learning Service many behaviors, such as: "Play the music loud usually, soft when PowerPoint Application A is running, and really soft when PowerPoint Application B is running." We generated training examples every 10 seconds, and started and stopped applications while adjusting the MP3 Application's volume to correspond to the different applications and different users within the environment roughly every minute for five minutes. We were able to learn this behavior easily using SNoW's default Naive Bayes, Winnow, and Perceptron algorithms. Thus, the agent was able to interact with the Learning Service and automate the environment according to the demonstrated behavior as applications were started and stopped in the environment. We augmented this experiment with the use of fingerprint scanners and RF tags to detect user presence, and more complicated concepts were learned, such as "Play music softly when a presentation is occurring, loudly otherwise, except if Professor Campbell is detected within the room."

We occasionally experienced poor results when the algorithms employed by SNoW did not produce the necessary approximation functions to reflect the demonstrated behavior. Often this was a result of insufficient amounts of training examples to compensate for noise in the contextual information. Additionally, the volume adjustments often lagged behind changes in context due to a system-level setting that caused a ten-second delay for notifications of exiting applications.

4.2. Experiment 2: Location Service

In the second experiment, we employed the Learning Service to automate both the volume of the MP3 Player and the X10 Application's lamp settings based on where in the environment a user is located. This experiment is based on our motivational scenario: both light settings and volume settings will need to change to provide appropriate light and sound for users performing different activities and moving throughout the environment.

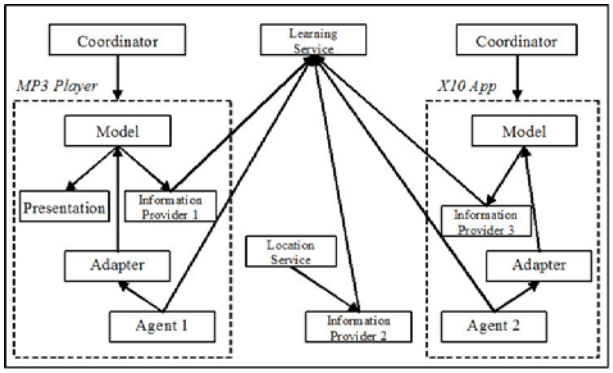

Figure 9. Experiment 2 configuration

Figure 9 shows the configuration of the Machine Learning system for this experiment. Information providers deliver the volume of the MP3 Player, entrances of users into four regions associated with four different plasma displays, and the state of lamps controlled by the X10 system, which controls electronic appliances and lights by passing messages over an environment's electrical system. During the training phase, a controller resides on each of the four plasma displays with bridges to the MP3 Player and X10

Application, allowing users to set the volume and lighting as they move about the environment. During the automation phase, agents query the variable for the volume of the MP3 Player Application and the X10 Application's lamp's status every second and change the lamp and volume settings according to the result.

Using this setup, a user moves among the different regions, setting the lighting and volume to different levels based on where she is in the environment. The user stands in each region for a small period of time after she change the settings to simulate more realistic usage conditions. Training examples are generated every second for roughly one or two minutes. We employ SNoW using Naive Bayes to learn approximations based on the training examples we generated. Then, we launch agents that automate the environment as the user walks between plasma displays again.

In the worst case, the environment is only able to automate some of the preferences that the user demonstrated. Depending on the noise in the training examples, which is strongly influenced by the accuracy of the location system, usually anywhere between half and all of the preferences are learned and automated. During automation the environment is much more responsive than in the first experiment as long as users are accurately detected within the different regions of the environment, as changes in location are propagated nearly instantaneously, unlike closing applications, with their ten-second delays. Successful automation often occurred in a cascading pattern; first the lights would change as the user migrates and the music would change after the light change had registered in the state representation, as the lights being on or off were good indicators of a specific volume level when combined with the user's location. Occasionally, when our system failed to automate the light settings appropriately, volume settings would also not be appropriately automated without first manually controlling the lights.

4.3. Discussion of Results

The experiments we employed were designed to quickly demonstrate the ability of the learning system after only a few minutes of training. The experiments were generally successful with such short training periods because of the small feature space in each experiment. As with all supervised classification problems, we expect that as the feature space increases, a longer training period will be required to learn the same automation policies. Additionally, successful experiments also depended on examples in the testing data being somewhat represented in the training data's examples.

Our choice of algorithm in the learning phase of the experiments had only a small impact on the system's efficacy in automating the environment, but some fine tuning was required. For example, the Naive Bayes algorithm required a high smoothing parameter value for unseen attributes to compensate for new contextual information not present during training (e.g., a new user registering within the environment during the automation phase). This might make Naive Bayes a poor algorithm choice if, through the somewhat random usage of the environment, many previously unseen features such as this may appear during testing. However, we are reluctant to characterize the data from a fully deployed pervasive computing environment based on our experiments. Many diverse aspects of the environment will need to be automated using the Learning Service to get a sense for what characteristics exist in the training examples and what specialized algorithms might be best suited to handle them.

The batch-based learning approach, where all the training examples are generated and then a classifier is trained, was appropriate for these experiments because we were learning unique policies demonstrated in each of the small windows of training. SNoW has support for training also over testing data, but for our experiments this would have just reinforced whatever was already being automated. In future deployments, continuous observation of users' interactions with the environment and real-time refinements of the learned automation policies will be desirable. There are additional system and modeling considerations involved with discerning automated actions from user actions in our approach, such as tracking through applications and Gaia bridges whether an agent or user originated some action, which we discuss in our future work section.

5. Related Work

In this section, we discuss how our Learning System relates to other work in the field. In particular, we look at the handful of similar systems designed to learn and automate user preferences within an everyday environment, and works in the fields of context-based computing and activity recognition.

5.1. Automating Environments by Observed Usage

There are a handful of similar systems that automate their environments based on the observed usage. The Neural Network House [12] controls operations of a house's air, heating, cooling, and ventilation systems by observing users' control decisions and sensor readings throughout the house. The system employs neural networks to learn how to control environmental conditions within the house based on the needs of users while reducing energy usage.

The iDorm system [4] consists of many different sensing devices and controllable devices embedded within a dormitory room. When the user controls one of the devices within

the environment, the state of the environment immediately prior to the action is recorded. A rule extraction system based on fuzzy logic is employed to find appropriate automation policies. These policies are then automated within the environment to reduce the amount of user intervention.

In Project Oxygen, a system observes the usage of an environment and utilizes a human tracking system to create "activity zones" where users perform certain types of actions using clustering techniques [9]. The system relies on the user to label zones of interest and associate appropriate actions to take when events occur in each zone.

In the MavHome Project [20], the environment is represented using a Hierarchical Hidden Markov Model, and a reinforcement learning algorithm is employed to predict environmental preferences based on sensors within the environment. Desired actions are proposed for the control of lights within the environment primarily based on motion detection sensors, and, if the actions are within the bounds of acceptable safety and security policies, they are invoked within the environment. Using this system to automate the environment, fewer user interactions with lighting controls are required while power consumption is reduced.

Our work is unique in that rather than engineering a learning system to handle specific controls and sensors, we maintain Gaia's operating system approach of providing generic support for additional expansion. Within the Gaia framework, application developers provide the necessary functionality to accompany a new source of contextual information or a new invokable control by encapsulating it as an application. The application can then be deployed and will be used by the learning service in any new Gaia environment without requiring an expert to connect and configure it. This is comparable to how many different input and output devices can be used without modification to existing programs in a traditional operating system. It is important for users who want to be able to add new devices to their environments to be able to do so without requiring an expert's assistance.

5.2. Context-based Computing

Research in context-based computing is primarily focused on gathering raw contextual information from sensing devices, inferring higher-level contextual information from this raw contextual data, and providing contextual information to context consumers. The synthesized context can be based on provided logic rules. Dey et al. [2] created a context toolkit that provides this functionality to enable context-aware applications. Our work is separate from context infrastructures. The Gaia Learning Service learns to invoke changes in the environment, whereas context infrastructures focus on providing more complex contextual information as application input for programmers to use. In our work, we use a direct mapping between low-level contextual data and features for learning, enabling users rather than programmers to have control over their environments responses to changes in context.

5.3. Activity Recognition

Much of the machine learning research in pervasive computing environments is focused on activity recognition [13, 8, 3]. A high-level description of the room's usage or the activities of users is desired as a function of all the contextual information provided by sensors within the environment. Unlike the Gaia Learning System, a major limitation of activity recognition is obtaining training data labeled with user-supplied activity labels [7]. Though clustering approaches and background knowledge can possibly reduce or eliminate the need for exterior supervision in some domains [11, 14], generally users or observers must supply these labels in a manner that is not a natural part of their usage of the environment. While knowing the high-level classification of the room's usage can be very useful for automating the environment, the premise of our research is that we can learn to automate the environment based on what is observable without external labeling.

6. Conclusions and Future Directions

All the decisions a user makes when interacting with the world around him or her is some function of the global environment information (including the neuron activity within the user's brain). When the function can be approximated by the observable environment information, it becomes possible to use machine learning techniques to learn the approximation and later use it to automate the environment. Pervasive computing research provides mechanisms for easily integrating many different sensors and interaction devices into the observable environment.

In this paper, we used our pervasive computing environment, Gaia, to develop a general state representation for the environment that accepts many sources of contextual information and supports abstract features using entities and entity referencing. We extended existing concepts from within the pervasive computing infrastructure to provide contextual information from many sources within the environment. We employed the SNoW Machine Learning architecture to learn approximations for each contextual attributes as a function of all other contextual attributes in our environment representation. We used these approximations to create predictions for different contextual characteristics under the current state of the environment, and automated the environment based on the predictions. Finally, we demonstrated how our Learning Service can be used in two experiments motivated from our original scenario.

Our Learning Service provides a more practical, more natural alternative to the existing approaches for specifying automation policies. Unlike manual rule specification tools, we employ machine learning techniques that scale better for policies that are complicated combinations of contextual information and for environments with an abundance of contextual information. Unlike some of the other machine learning approaches, we use the user's natural interactions with the pervasive computing environment rather than requiring external labeling to learn automation policies, and our system is designed to allow the entire observable context of the environment and all the available actions to be used in automation policies. Additionally, our system allows the user to add additional features and actions encapsulated within developed application to the automation system without requiring an expert. Our approach enables users to more easily automate the behaviors of pervasive computing environments to reflect personal preferences, allowing more of the utility that pervasive computing research promises to be realized by the end-user. While our results are already very promising, we expect the comparative benefits of this approach to grow as the contextual awareness of pervasive computing environments continues to increase.

6.1. Online Learning and Causal Models

One area of our work that we are planning to address is our use of batch-based learning, where a number of training examples are collected and then given to a machine learning system to learn from. Batch-based learning imposes an unnatural mode of user interaction with the pervasive computing environment. More desirable for an interactive environment is to use online learning, where a system learns continually as the user changes the environment in response to contextual changes, and also responds to the automated decisions of agents. However, at the system level the Gaia infrastructure does not distinguish between the actions invoked by users and those invoked automatically by agents. Without being able to tell the difference between these two types of actions, the learning system would strongly reinforce whatever behaviors it learned earlier that are driving the decisions of agents.

Additionally, without knowing what effects an action has on the context of the environment, learning algorithms will discover the correlations between the action and its effects, which is useless for prediction. One of the ways that other learning systems avoid this problem is by selecting controls and sensors so that the effects of controls are not directly detected, or, when they are, they are specifically ignored. This solution is not practical in enabling the user to have maximal control over the environment without requiring an expert to perform feature selection for every controllable action. Another way other learning systems avoid the problem is by only learning when a user overrides control, but this approach ignores that not taking control can be usefully interpreted as implicit approval.

Support could be added within the Gaia infrastructure to track the causation of actions and differentiate those caused by agents from those caused by users. Additionally, causal models of actions and their effects on contextual information could be automatically created and used to distinguish between changes in the environment's contextual information that are effects of some action and changes that might indicate a triggering situation for that action. The machine learning system could then be modified to only learn user-imposed changes in the environment as a function of the environment's context that is not an effect of the action. Additionally, introducing a means for overriding the agent-enforced preferences would temporarily allow user decisions to stand without being automatically changed. This would also make room for learning individual automation preferences for each user when there are many users in a single environment.

References

[1] A. J. Carlson, C. M. Cumby, J. L. Rosen, and D. Roth. SNoW user guide. Technical Report 2101, University of Illinois, Urbana, Illinois, 1999.

[2] A. Dey, G. Abowd, and D. Salber. A context-based infrastructure for smart environments. In *Proceedings of the 1st International Workshop on Managing Interactions in Smart Environments (MANSE '99)*, pages 114–128, Dublin, Ireland, December 1999.

[3] A. K. Dey, R. Hamid, C. Beckmann, I. Li, and D. Hsu. a cappella: Programming by demonstration of context-aware applications. In *Proceedings of the 2004 Conference on Human Factors in Computing Systems*, pages 33–40, Vienna, Austria, April 2004.

[4] H. Hagras, V. Callaghan, M. Colley, G. Clarke, A. Pounds-Cornish, and H. Duman. Creating an ambient-intelligence environment using embedded agents. *IEEE Intelligent Systems*, 19(6):12–20, 2004.

[5] C. Hess, F. Ballesteros, and R. Campbell. An adaptable distributed file service. In *Proceedings of the ECOOP PhD Workshop on Object Oriented Systems (PHDOOS'00)*, June 2000.

[6] R. Ierusalimschy, L. H. de Figueiredo, and W. C. Filho. Lua — an extensible extension language. *Software: Practice and Experience*, 26(6):635–652, 1996.

[7] S. S. Intille, L. Bao, E. M. Tapia, and J. Rondoni. Acquiring in situ training data for context-aware ubiquitous computing applications. In *Proceedings of the 2004 Conference on Human Factors in Computing Systems*, pages 1–8, Vienna, Austria, April 2004.

[8] N. Kern and B. Schiele. Multi-sensor activity context detection for wearable computing. In *European Symposium on Ambient Intelligence*, pages 220–232, Eindhoven, The Netherlands, November 2003.

[9] K. Koile, K. Tollmar, D. Demirdjian, H. Shrobe, and T. Darrell. Activity zones for context-aware computing. In *Proceedings of UBICOMP 2003: The Fifth International Conference on Ubiquitous Computing*, pages 90–106, Seattle, WA, October 2003.

[10] G. Krasner and S. Pope. A description of the model-view-controller user interface paradigm in the smalltalk-80 system. *Journal of Object Oriented Programming*, 1(3):26–49, 1988.

[11] A. Krause, A. Smailagic, D. Siewiorek, and J. Farringdon. Unsupervised, dynamic identification of physiological and activity context in wearable computing. In *Proceedings of the Seventh IEEE International Symposium on Wearable Computers*, pages 88–97, White Plains, NY, October 2003.

[12] M. Mozer. The neural network house: An environment that adapts to its inhabitants. In *Proceedings of the AAAI 1998 Spring Symposium on Intelligent Environments*, pages 110–114, Menlo Park, CA, March 1998.

[13] N. Oliver, E. Horvitz, and A. Garg. Layered representations for human activity recognition. In *Proceedings of the International Conference on Multimodal Interfaces*, pages 3–8, Pittsburgh, PA, October 2002.

[14] D. J. Patterson, L. Liao, D. Fox, and H. Kautz. Inferring High-Level Behavior from Low-Level Sensors. In *Proceedings of UBICOMP 2003: The Fifth International Conference on Ubiquitous Computing*, volume LNCS 2864, pages 73–89. Springer-Verlag, October 2003.

[15] N. Rodriguez, R. Ierusalimschy, and R. Cerqueira. Dynamic configuration with corba components. In *CDS '98: Proceedings of the International Conference on Configurable Distributed Systems*, pages 27–34, May 1998.

[16] M. Roman and R. H. Campbell. A middleware-based application framework for active space applications. In *ACM/IFIP/USENIX International Middleware Conference*, pages 433–454, Rio de Janeiro, Brazil, June 2003.

[17] M. Roman, B. Ziebart, and R. H. Campbell. Dynamic Application Composition: Customizing the Behavior of an Active Space . In *IEEE International Conference on Pervasive Computing and Communication*, pages 169–176, Dallas-Fort Worth, Texas, March 2003.

[18] B. Schilit, N. Adams, and R. Want. Context-aware computing applications. In *IEEE Workshop on Mobile Computing Systems and Applications*, pages 85–90, Santa Cruz, CA, US, December 1994.

[19] M. Weiser. The computer for the 21st century. *Scientific American*, 265(3):66–75, January 1991.

[20] G. M. Youngblood, L. B. Holder, and D. J. Cook. Managing adaptive versatile environments. In *IEEE International Conference on Pervasive Computing and Communications*, pages 351–360, Kauai Island, HI, March 2005.