

# Inhabitant Guidance of Smart Environments

Parisa Rashidi<sup>1</sup>, G. Michael Youngblood<sup>2</sup>, Diane J. Cook<sup>1</sup>, and Sajal K. Das<sup>3</sup>

<sup>1</sup>Washington State University, Pullman, WA 99163 USA  
{prashidi,cook}@eecs.wsu.edu

<sup>2</sup>University of North Carolina at Charlotte, Charlotte, NC 28221 USA  
youngbld@uncc.edu

<sup>3</sup>University of Texas at Arlington, Arlington, TX 76009 USA  
das@cse.uta.edu

**Abstract.** With the convergence of technologies in artificial intelligence, human-computer interfaces, and pervasive computing, the idea of a “smart environment” is becoming a reality. While we all would like the benefits of an environment that automates many of our daily tasks, a smart environment that makes the wrong decisions can quickly become annoying. In this paper, we describe a simulation tool that can be used to visualize activity data in a smart home, play through proposed automation schemes, and ultimately provide guidance to automating the smart environment. We describe how automation policies can adapt to resident feedback, and demonstrate the ideas in the context of the MavHome smart home.

## 1 Introduction

Since the beginning, people have lived in places that provide shelter, basic comfort, and support, but as society and technology advance there is a growing interest in improving the intelligence of the environments in which we live and work. A “smart environment” acquires and applies knowledge about the physical setting and its residents, in order to improve their experience in the setting. A smart environment can be treated as an intelligent agent, which perceives the state of the environment using sensors and acts upon the environment using device controllers in a way that can optimize a number of different goals including maximizing comfort of the inhabitants, minimizing the consumption of resources, and maintaining safety of the environment and its inhabitants.

As the need for automating these personal environments grows, so does the number of researchers investigating this topic. Some researchers are targeting individual devices that provide a useful function, such as programmable appliances. While these devices are novel and useful for limited tasks, they typically do not consider the bigger picture of interacting with the rest of the environment. Others design interactive conference rooms, offices, kiosks, and furniture with seamless integration between heterogeneous devices

and multiple user applications in order to facilitate collaborative work environments. Ideas for recognizing environment resident activities [3], for planning environment reminders [5], and for reacting to hazardous situations [4] have been discussed.

Smart environments have the potential to aid people with mental and physical limitations, to provide resource conservation, and to make our lives more comfortable and productive. The main hindrance to realize this potential is the ease with which smart environment technology can be integrated into the lifestyle of its residents. Our vision is to design a smart environment that adapts to its residents. With our approach, the resident plays a critical role in guiding the environment's automation policy.

In this paper we describe an important step toward that goal: design of the ResiSim simulation environment. This environment can be used to visualize daily resident activities and proposed automation schemes. We will explain how it can ultimately be used as a tool for capturing user feedback, and will describe how such feedback can be incorporated into the automation model. We will describe this approach in the context of the MavHome smart home environment.

## 2 The MavHome Smart Home Environment

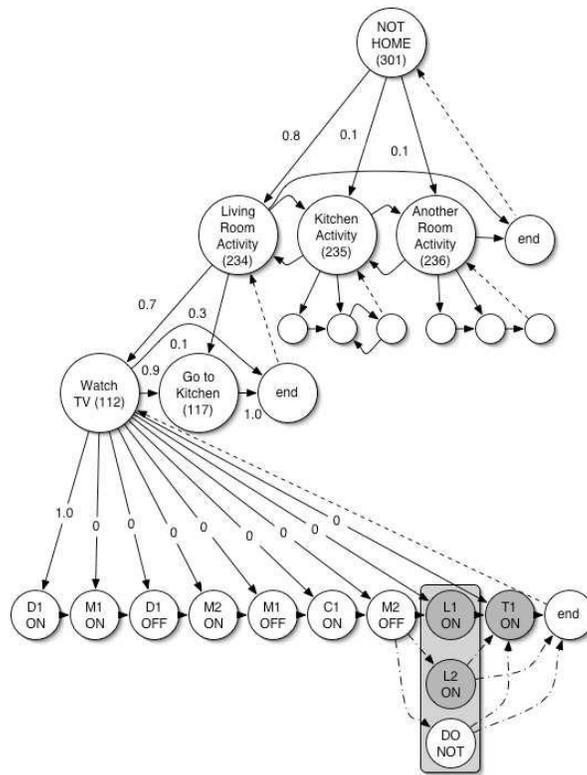


**Fig. 1.** The MavLab (left) and MavPad (right) smart environments.

The MavHome project treats the smart home environment as an intelligent agent. The home agent perceives the environment using sensors and acts upon the environment using powerline controllers. The MavHome algorithms have been tested in two physical environments, and MavLab workplace environment and the MavPad on-campus apartment. The MavLab (Figure 1 left) contains work areas, cubicles, a break area, a lounge, and a conference room, and is equipped with 54 X10 controllers and sensors for light, temperature, humidity, motion, door usage, and seat usage. The MavPad apartment (Figure 1 right) hosts a full-time volunteer resident and offers additional sensing for leak detection, vent position, smoke detection, CO detection, and window status.

To automate our smart environment, we collect observations of manual resident activities and interactions with the environment. We mine sequential patterns from this data using a sequence mining algorithm, ED [2]. ED identifies sequences of resident activities that yield a high value using the Minimum Description Length (MDL) principle. The MDL measure prefers that are frequent, long, and periodic (occur at regular intervals).

Next, we predict the resident's upcoming action using a text-based prediction algorithm, ALZ [1]. ALZ parses the history of observed events into substrings and stores them with frequency information in a tree. Relative frequencies from different size contexts (phrase sizes) are combined to generate the probability of each possible action being the next one to occur. The most-likely action is the one reported by ALZ.



**Fig. 2.** An example hierarchical Model created from observed resident activities.

Finally, a hierarchical Markov model is automatically created from historical event data. Our PropHeT decision learner uses nodes at the lowest level of the hierarchy to

represent raw event information, and uses nodes at the higher levels of the hierarchy to represent ED-discovered abstract sequences (see Figure 2). ProPHeT use reinforcement learning to generate a policy for this model that will control the environment.

We have implemented a number of case studies in the MavLab and MavPad. The MavLab has been used to automate the activities of a group of six students and the MavPad hosted a succession of three student volunteers. In our case studies, this approach to automation was successful in reducing the number of manual interactions by approximately 75% [7].

### 3 Simulation of Smart Environments

Simulation is an important tool for performing evaluation and research in many areas. This is particularly true for smart environments, where simulation plays many key roles. First, simulation tools can be used to visualize resident activities, which will in turn provide insights on patterns that can be automated and trends that need to be monitored.

We designed an algorithm, ResiSim, that simulates smart environments as a discrete event simulation. Each object is a self-contained, accurate simulation of an item in the environment and has the ability to be replaced by actual objects, bridging the gap between reality and virtual reality. As in most real-world environments, new objects can be introduced at any time and existing objects can be removed.



Fig. 3. Simulation object types in ResiSim.

ResiSim is a Zero Configuration distributed simulator. As such, objects can be of any size and functionality, and are categorized as static, dynamic, simulation, observation, and interface. Static objects simply exist and do not move. Dynamic objects can move in the environment or change state. Simulation objects provide mathematical support to coordinate interactions between other objects, and observation objects provide a view into the simulation. Interface objects allow either users or other external entities to interact with the simulation. While each object contains attributes and maintains a state, it may also provide functionality in the form of operators. To run a simulation, all that is necessary is to start all of the objects. The static objects make up the environment while the dynamic objects are the key actors. The simulation objects determine the effects of the dynamic objects on each other, causing state changes as necessary. The simulation can be viewed through an observation object that may appropriately display the state of the simulation. User interaction or external information may be channeled through an interface object. Updating is usually event driven, but other types of simulations could be established depending on the core programming of the objects and their defined attributes and operations.

ResiSim consists of three main parts: logical proxy objects, core simulation objects, and user interface objects. Logical proxies are ZeroConf-enabled object processes that are distributed among the available computing assets on the network and represent a physical object (e.g., a lamp or a chair) in the real world. Logical proxies in our work are usually static objects, but could be dynamic if they move around or change state in the environment. Our logical proxies usually take one of two forms — physical or simulated. Physical logical proxies are used to bridge the gap between reality and virtual reality by projecting a logical interface to a real-world object into the simulation environment so that the simulator can actually control the real-world object. For example, we can allow some lights in the real-world to be turned on and off through simulation actions (e.g., a virtual inhabitant can turn on real-world lights). Simulated logical proxies are virtual reality objects that simulate the behavior of the real-world objects they represent.

The core simulation objects include ZeroConf-enabled virtual inhabitant dynamic objects, simulation objects, interface objects, and an observation object. Figure 4 (left) shows an observation object view of the MavLab simulation and Figure 4 (right) shows an interface panel which allows selection of the simulated objects. In our implementation, we aggregate the core simulation objects into the ResiSim Server component. We call this a server because from the simulation core we can serve to clients outside our link-local addresses through IP connections. If the clients are within the link-local network then they participate as ZeroConf objects; otherwise, they must be configured to locate the server on the Internet. The server provides a collecting point for all of the necessary components for a complete simulation; however, all of the constituent components are independent ZeroConf objects. This approach allows for easy dynamic changes in the simulation where any object can be added or removed to the simulation. Removed objects notify ResiSim of their removal through ZeroConf, allowing interacting objects to accommodate the change. Added objects notify ResiSim of their availability and service through

ZeroConf, allowing other objects to seek interactions with them. These components comprise a complete Zero Configuration distributed simulation environment.

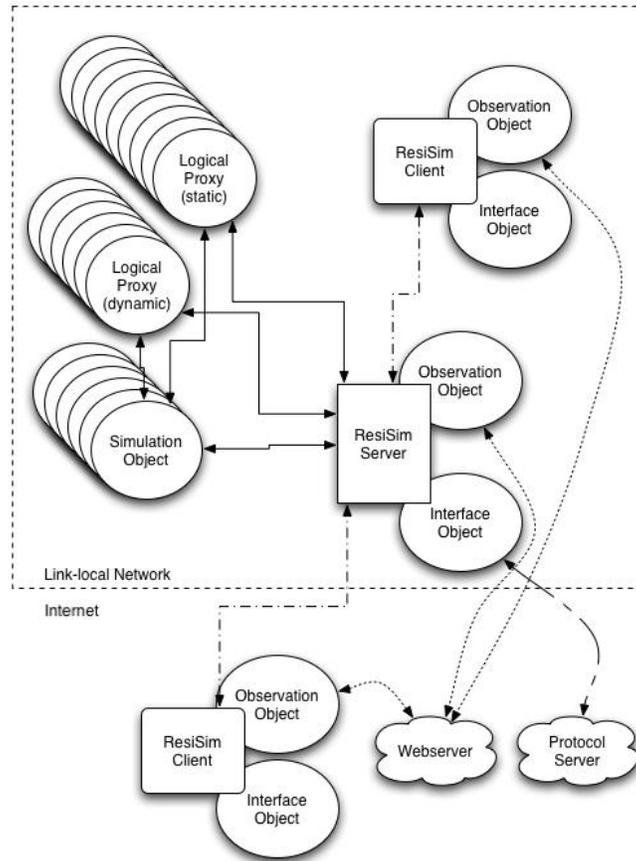
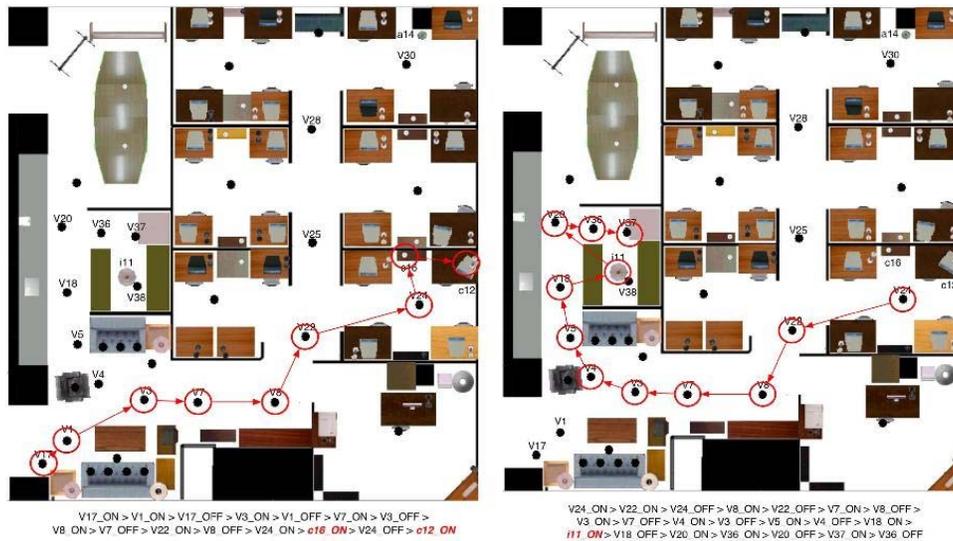


Fig. 4. ResiSim architecture.

The ResiSim software architecture is shown in Figure 4. We utilize both real-world and simulated logical proxy objects in our simulation, which allows the decision making function to control either virtual objects in a simulation environment or physical objects in a real-world environment in an identical fashion. This way, the learned policy can be simulated before the policy goes into effect in the physical environment. We have complete 2D and 3D models of the MavLab (containing of 139 modeled objects) and the MavPad (containing 111 modeled objects). These simulations are used to visualize resident activities and learned control policies, which we will discuss next.

## 4 Guiding a Smart Environment

Our initial studies indicate that machine learning technologies can be used to model and predict inhabitant activities, and that a policy can be learned using this information to automate a smart environment. However, what is not investigated here, or in any smart environment project, is how beneficial and pleasing the automation is to the inhabitant. If the environment makes a wrong automation decision, it will be at best an annoyance and at worst a hindrance to accomplishing everyday tasks.



**Fig. 5.** Visualized MavLab patterns of lab entry to desk (left) and going on break (right). Sensors starting with “V” are motion or door sensors, the rest are X10 powerline controllers.

We are investigating techniques for integrating inhabitant feedback and guidance into a smart environment. To do this, the agent's reward signal will now combine feedback from the environment and from the human inhabitant. Designing environments that learn by interacting with everyday people is a neglected topic, although some researchers have investigated human interaction with robots as a teaching mechanism].

Inhabitant training of the environment will be based on reactive to inhabitant behavior and on proactively allowing the inhabitant to guide future policy based on simulation. For each approach, we will also need a machine learning algorithm that not only learns an initial model from data, but also refines the model to encompass new actions and alternatives as suggested by the inhabitant.

#### **4.1. Reaction to Inhabitant Feedback**

If a smart environment is designed for life-long use, it must have the ability to react to feedback from the inhabitant and adapt to changes in inhabitant lifestyle. An example of reacting to inhabitant feedback is the immediate reversal of an automation action. If the inhabitant immediately turns a light back off that the environment turned on, this constitutes a negative reinforcement for the learner.

A manually-constructed hierarchy will be brittle in the face of such changes. However, our data-driven approach should be able to accommodate these changes, making incremental modifications to the model when appropriate and triggering construction of a new model when needed. Incremental modifications of ProPHeT's hierarchical model will occur based on inhabitant interaction. If Event E is countermanded (reversed within a time threshold from the point at which it was automated) or replaced by an alternative inhabitant action (he prefers light a16 to be on instead of a14), two parallel states are added to the model. The first is a "NO ACT" state. This allows ProPHeT to leave one of the automated actions out given sufficient inhabitant feedback. The second is a state corresponding to the alternative action (in our example, a "a16 ON" state). As shown in Figure 2, these new states connect to all possible downstream states in the Markov chain. Through experience ProPHeT can learn what the appropriate connection should be.

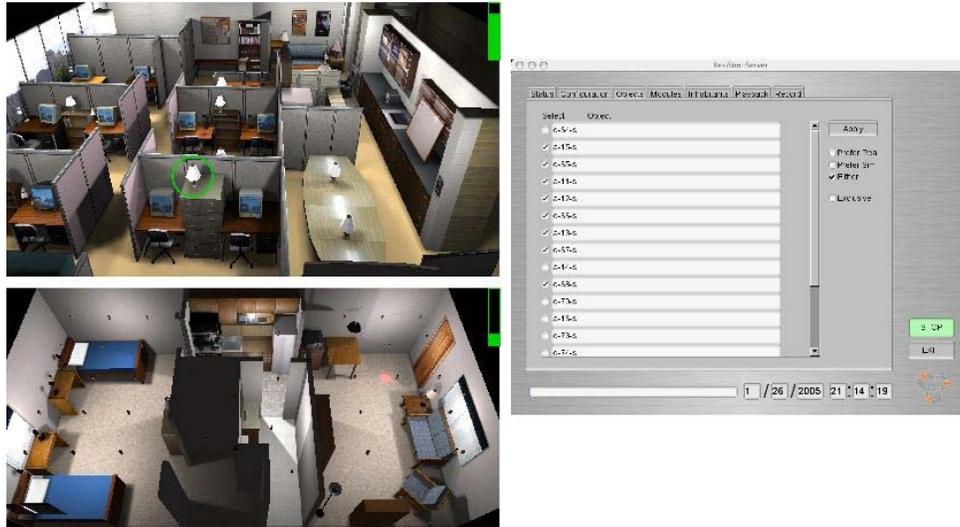
We envision that over time, some of the states in the model may not be needed because they no longer reflect events that actually occur. In order for state space reduction to occur, we will give episodes and individual states a time-to-live counter that is refreshed each time a given part of the hierarchical model is traversed. Unused sections of the HPOMDP will be pruned for increased efficiency. This assists in removing unused episodes and clutter inserted from feedback learning that may have been placed without sufficient continued reinforcement. Keeping the model as clean and dynamic as possible promotes fast policy convergence.

Finally, if both the performance of the ProPHeT model degrades steadily over time, this may be an indicator of a drift, or a steady change in inhabitant actions. When this occurs over a period of time, ProPHeT will initiate learning of a new model based on recent data (after the performance started to change). If the new model performs better on recent historic data, a system reboot will be initiated to use the new model.

#### **4.2. Inhabitant Guidance of the Environment**

Thomaz and Brezeale [6] show that robots performed better when the human trainer moved beyond reacting to robot actions and guided to the robot to future beneficial tasks. environment. As shown in Figure 5, ResiSim can be run at a real-world pace or sped up to highlight daily activities. Simulation of inhabitant activities in our earlier experiments provides a great deal of information. In the lab setting we can easily observe when the inhabitant arrives, works on the computer, takes a break, interacts with other people in the lab, and leaves. In the apartment setting the observer can tell, for example, when the

inhabitant is cooking, eating, taking a shower, watching television, working, sleeping restfully, and stirring in their sleep.



**Fig. 6.** 3D ResiSim views of the MavLab (top left) and MavPad (bottom left). The reinforcement bars are shown in the upper right of each display, and the lamp object is highlighted in the MavLab. A corresponding interface panel for the lamp appears (right) when the object is selected.

Using the ResiSim simulator, inhabitants can view past actions and automations. They can also view a simulation of a new policy, by watching the simulated actions play out based on projected inhabitant activities. A feedback bar will be shown on the display (see Figure 6), and the viewer can slide the bar to the appropriate level. The reward signal  $r$  is a scalar value in the range  $[-1,1]$ . The user can at any point in the simulation click on the interface panel in order to provide reward to the system.

In addition, passing the mouse over individual objects highlights the objects and brings up the corresponding panel. The user can select the desired status of the device (e.g., turn lamp 16 on or off) and a corresponding reward, inputting both pieces of information to the system with a single click. This will guide ProPHeT to introduce a new action at the given point in the policy that has a corresponding reward. For example, a worker may like the lamp on their desk to be illuminated when they first enter the lab. ProPHeT may not include this option in the current model because the inhabitant typically does not perform this activity. However, by highlighting the light object and clicking on a reward, the new possible action will be added to the model and given the corresponding reinforcement.

## 5. Conclusions and Future Work

In this paper we discussed a distributed simulator that provides visualization of smart environment resident activities and learned control policies. This simulator will ultimately allow users to provide reactive feedback and proactive guidance to the environment. We anticipate that user guidance of our smart environments will increase satisfaction with automation assistance and promote more wide-spread use of smart environment technologies. Our user interface will be ported to a PDA that the inhabitant can use to provide guidance to the system anywhere, anytime. The PDA interface will also allow the inhabitant to monitor and control the environment remotely. The algorithms will be tested in a smart lab on campus, in a smart apartment on campus with volunteer inhabitants, in a smart apartment of a volunteer elder adult, and in an assisted care facility.

## References

1. K. Gopalratnam and D. Cook. Online sequential prediction via incremental parsing: The Active LeZi Algorithm. *IEEE Intelligent Systems*, 22(1) (2007).
2. E.O. Heierman and D. Cook. Improving home automation by discovering regularly occurring device usage patterns. *Proceedings of the International Conference on Data Mining* (2003).
3. S. Luhr. Recognition of emergent human behavior in a smart home: A data mining approach. *Journal of Pervasive and Mobile Computing*, 3 (2007).
4. S. Moncrieff. Multi-model emotive computing in a smart house environment. *Journal of Pervasive and Mobile Computing*, 3 (2007).
5. R. Simpson, D. Schreckenghost, E.F. LoPresti, and N. Kirsch. Plans and planning in smart homes. In J. Augusto and C. Nugent (eds.), *AI and Smart Homes*, Springer Verlag (2006).
6. A.L. Thomaz and C. Brezeal. Reinforcement learning with human teachers: Evidence of feedback and guidance with implications for learning performance. *Proceedings of the National Conference on Artificial Intelligence* (2006).
7. G. M. Youngblood and D. J. Cook. Data mining for hierarchical model creation. *IEEE Transactions on Systems, Man, and Cybernetics, Part C* (2007).