The Role of Prediction Algorithms in the MavHome Smart Home Architecture *

Sajal K. Das, Diane J. Cook, Amiya Bhattacharya,
Edwin O. Heierman, III, and Tze-Yun Lin

Department of Computer Science Engineering
University of Texas at Arlington
Box 19015, Arlington, TX 76019-0015
Email: {das,cook,bhatt,heierman,tylin}@cse.uta.edu

Abstract

The goal of the MavHome (**M**anaging **A**n Intelligent **V**ersatile **Home**) project is to create a home that acts as a rational agent. The agent seeks to maximize inhabitant comfort and minimize operation cost. In order to achieve these goals, the agent must be able to predict the mobility patterns and device usages of the inhabitants. In this paper, we introduce the MavHome project and its underlying architecture. The role of prediction algorithms within the architecture is discussed, and three prediction algorithms which are central to the home operations are presented. We demonstrate the effectiveness of these algorithms on synthetic and/or actual smart home data.

1 Introduction

The **MavHome** smart home project is a multi-disciplinary research project at the University of Texas at Arlington focused on the creation of an intelligent and versatile home

*This work was supported by National Science Foundation grant IIS-0121297.

environment. Our goal is to create a home that acts as a rational agent, perceiving the state of the home through sensors and acting upon the environment through effectors (in this case, device controllers). The agent acts in a way to maximize its goal, which is a function that maximizes comfort and productivity of its inhabitants and minimizes operation cost. In order to achieve these goals, the house must be able to reason about and adapt to its inhabitants. In particular, a smart home agent must be able to accurately predict mobility and other activities of its inhabitants. Using these predictions the home can accurately route messages and multimedia information, and can automate activities that would otherwise be manually performed by the inhabitants.

MavHome operations can be characterized by the following scenario. At 6:45am, MavHome turns up the heat because it has learned that the home needs 15 minutes to warm to optimal temperature for waking. The alarm goes off at 7:00, which signals the bedroom light to go on as well as the coffee maker in the kitchen. Bob steps into the bathroom and turns on the light. MavHome records this interaction, displays the morning news on the bathroom video screen, and turns on the shower. While Bob is shaving MavHome senses that Bob is two pounds over his ideal weight and adjusts Bob's suggested menu. When Bob finishes grooming, the bathroom light turns off while the kitchen light and menu/schedule display turns on, and the news program moves to the kitchen screen. During breakfast, Bob notices that the floor is dirty and requests the janitor robot to clean the house. When Bob leaves for work, MavHome secures the home, and starts the lawn sprinklers despite knowing the 70% predicted chance of rain. Later that day, MavHome places a grocery order for milk and cheese. When Bob arrives home, his grocery order has arrived and the hot tub is ready.

A number of capabilities are required for this scenario to occur, including data collection, activity prediction, wireless communication between multiple cooperating agents, and multimedia technologies. All of these capabilities must be seamlessly connected in a modular architecture. In this paper we present such an architecture that supports the MavHome smart home project, and then focus on several prediction capabilities that are needed to realize the types of activities described above.

In particular, machine learning techniques are required to predict inhabitant movement patterns, tasks, and typical interactions with the house, and to use that information in automating house decisions, routing information, and optimizing inhabitant comfort, security, and productivity. This paper introduces three such algorithms. The first predicts the mobility of the inhabitant using ideas from information theory. The second builds upon a sequence matching algorithm to predict inhabitant interactions with the smart home, and the third identifies significant patterns of inhabitant activity that could be automated by the home. We validate these algorithms on synthetic data sets as well as on activity data collected from real device usage histories.

2 MavHome Architecture

The MavHome architecture is a hierarchy of rational agents which cooperate to meet the goals of the overall home. Figure 1 shows the architecture of a MavHome agent. The technologies within each agent are separated into four cooperating layers. The **Decision** layer selects actions for the agent to execute based on information supplied from other layers. The **Information** layer gathers, stores, and generates knowledge useful for decision making. The **Communication** layer includes software to format and route information between agents, between users and the house, and between the house and external resources. The **Physical** layer contains the basic hardware within the house including individual devices, transducers, and network hardware. Because the architecture is hierarchical, the Physical layer may actually represent another agent in the hierarchy.

Perception is a bottom-up process. Sensors monitor the environment (e.g., lawn moisture level) and, if necessary, transmit the information to another agent through the Communication layer. The database records the information in the Information layer, updates its learned concepts and predictions accordingly, and alerts the Decision layer of the presence of new data. During action execution, information flows top down. The Decision layer selects an action (e.g., run the sprinklers) and relates the decision to the Information layer. After updating the database, the Communication layer routes the action to the appropriate effector to execute. If the effector is actually another agent, the agent receives the command through its effector as perceived information and must decide upon the best method of executing the desired action. A specialized interface agent provides interaction capabilities with users and

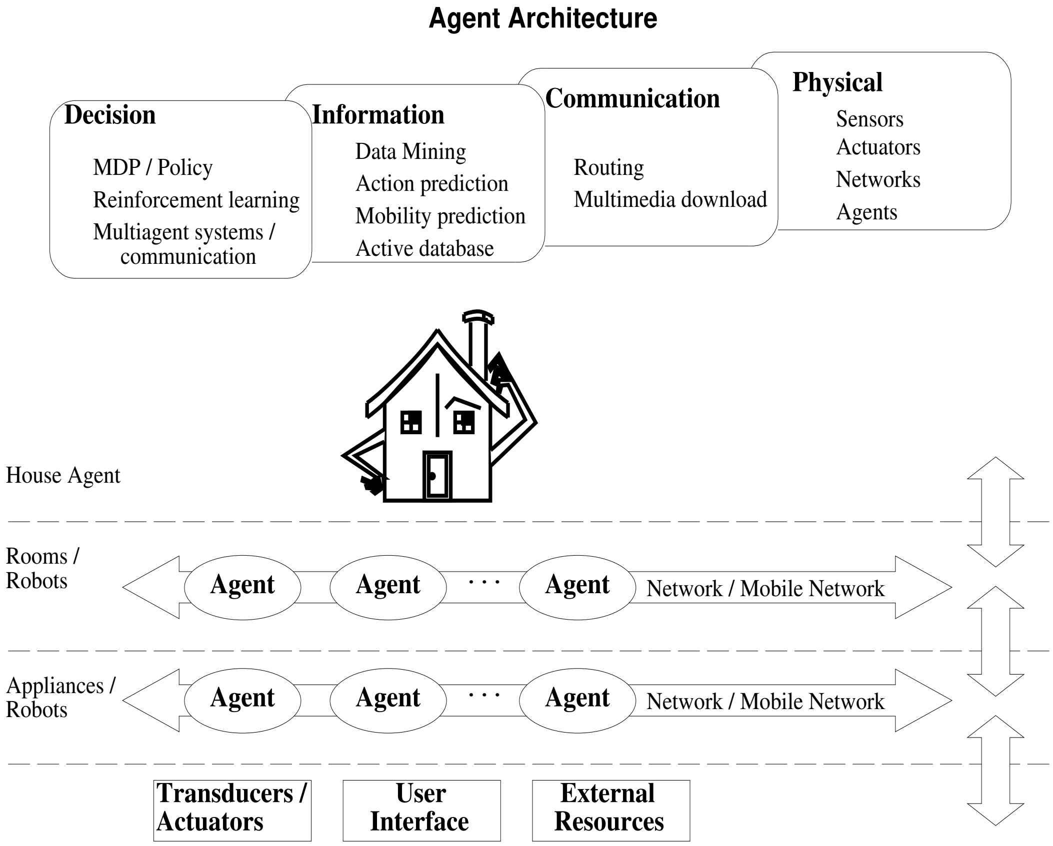

Figure 1: MavHome agent architecture.

with external resources such as the Internet. As shown in Figure 1, agents can communicate with parent/child agents or with other agents at the same level in the hierarchy.

Several smart home-related projects have been initiated by research labs. The Georgia Tech Aware Home and the MIT Intelligent Room include an impressive array of sensors to determine user locations and activities within an actual house. The Neural Network House at the University of Colorado Boulder employs a neural network to control heating, lighting, ventilation, and water temperature in a manner that minimizes operating cost. The interest of industrial labs in smart home and networked appliance technologies is evidenced by the creation of Jini, Bluetooth, and SIP (Session Initiation Protocol) standards, and by supporting technologies such as Xerox PARC's Zombie Board, Microsoft's Easy Living project, the Cisco Internet Home, and the Verizon Connected Family project. MavHome is unique in combining technologies from databases, artificial intelligence, mobile computing, robotics, and multimedia computing to create an entire smart home that acts as a rational agent.

3 Location Prediction

Seamless connectivity is an absolute necessity for designing an intelligent environments such as MavHome. To satisfy these connectivity requirements, the smart home needs to track down an inhabitant both inside and within surrounding areas. This is the primary scope of the *location management* problem in a smart home which exhibits some similarities with that of a typical wireless infrastructure network such as the land-mobile phone system. Wireless terminals are usually integrated in the sensors deployed in a smart home environment which are to be worn by the inhabitants.

The MavHome coverage area is partitioned into *zones* or *sectors*. Location management involves two types of activities. When MavHome needs to contact an inhabitant, the system initiates a search for the target terminal device by polling all zones where it can possibly be found. All terminals listen to the broadcast page message, and only the target sends a response. The paging process becomes partcularly inefficient if a large number of zones cover the entire smart environment. Unfortunately, due to inherent restrictions of the sensor technology such as infrared, these situations are sometimes unavoidable.

To control the location uncertainty of the inhabitant, MavHome must rely on time to time location update by the terminal device. This technique limits the search space for the next paging at the cost of a few location updates. Most paging algorithms can place high reliance on the latest update information, using the last known position and its surroundings as the most probable current positions [1, 2, 4]. Had probabilistic profile information been available for the inhabitant, it is also possible to modify the search space accordingly as proposed for some cellular phone systems [7].

We take a novel adaptive approach to the location management problem which is optimal in terms of both update and paging costs. The objective of our update scheme is to learn user mobility, endowing the paging mechanism with a predictive power which reduces average paging cost. Unlike earlier schemes, this provides an on-line algorithm for optimizing the paging problem and allows both the paging process and the update mechanism to be considered. machine learning methods are used to automate this process.

We represent a MavHome network (along with its neighborhood environment) by a bounded-degree, connected graph $G = (\vartheta, \varepsilon)$, where the node-set ϑ represents the zones and the edge-set ε represents the neighborhood (walls, hallways, etc.) between pairs of zones. Figure 2 shows a home floorplan with fifteen zones and the corresponding graph representation.

3.1 Movement history and mobility model

The real power of an adaptive algorithm comes from its ability to learn. As events unfold, a learning system observes the history to use it as a source of information. Referring to Bob's scenario, let us now look at Bob's movement history for one day within the house as shown in Figure 2. Table 1 shows the events which makes him cross zone boundaries. Let us deploy a pure movement-based scheme that generates an update whenever a zone boundary crossing is detected. The system thus receives a sequence of zone-ids as input yielding the movement history $mamcmrkdkdgoogdklrmcmamrlkdlrmamcmrkdkdgo\ldots$.

The *movement history* of a user is represented by a string "$v_1v_2v_3\ldots$" of symbols from the alphabet ϑ, where ϑ is the set of zones in the house and v_i denotes the zone-id reported by

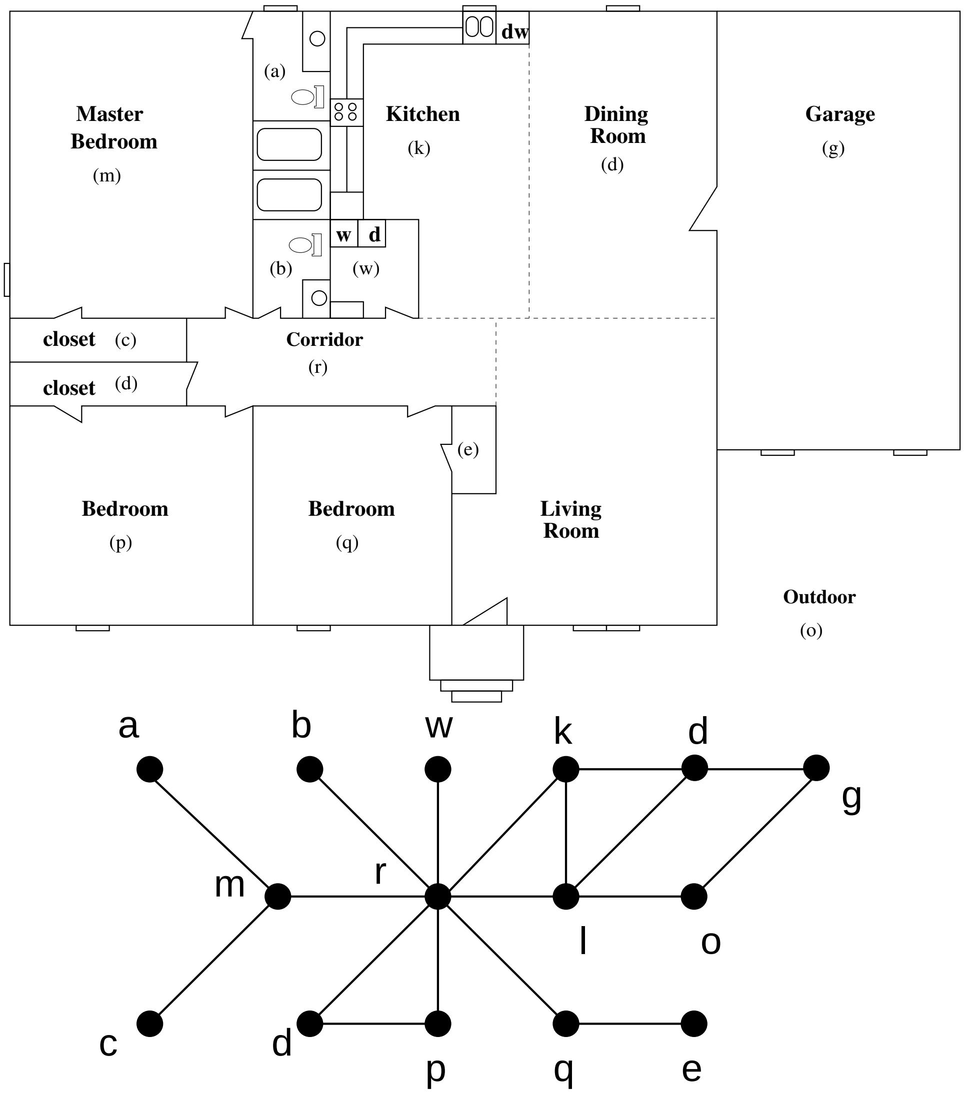

Figure 2: Graph model of a smart home floor plan.

Zone	Activity
m	Wake up in master bedroom
a	Go to attached bathroom
m	Back in bedroom
c	Change in closet
m	Back in bedroom
r	Out of bedroom
k	Go to kitchen to make breakfast
d	Go to dining room to eat
k	Back to the sink at kitchen
d	Walk to the garage through dining room
g	Start the car at garage
o	Drive away through outdoor area
o	Drive back through outdoor area
g	Back to garage
d	Enter through dining room
k	To kitchen for a snack and drink
l	To living room for watching TV
r	On the way to bedroom
m	In bedroom
c	To closet for changing
m	Back in bedroom
a	To attached bathroom
m	Back in bedroom
r	On the way for more TV watching
l	Back in living room
k	In kitchen to cook dinner
d	Eat dinner
l	Back for the Tonight Show
r	On the way to bedroom
m	In bed for the night
a	Wake up and go to bathroom
$\vdots$	

Table 1: Bob's movement history for a day

the ith update.

In contrast to a movement history, a mobility model is probabilistic and extends to the future. The tacit assumption is that an inhabitant's movement (favorite routes and habitual duration of stay) is merely a reflection of the patterns of his/her life, and those can be learned. Learning aids decision making when re-appearance of those patterns are detected. In other words, learning works because "history repeats itself".

The *mobility model* of a user is a stationary stochastic process $\mathcal{V} = \{V_i\}$, such that V_i assumes the value $v_i \in \vartheta$ in the event that the ith update reports the user in zone v_i. The joint distribution of any subsequence of V_is is invariant with respect to shifts in the time axis, i.e.,

$$\begin{aligned} & \Pr\left[V_1 = v_1, V_2 = v_2, \ldots, V_n = v_n\right] \qquad (1) \\ = \; & \Pr\left[V_{1+l} = v_1, V_{2+l} = v_2, \ldots, V_{n+l} = v_n\right] \end{aligned}$$

for every shift l and for all $v_i \in \vartheta$. The movement history is a trajectory or sample path of $\mathcal{V}$.

The goal of our algorithm is to construct a universal predictor or estimator for the user mobility model. Our proposed scheme creates a dictionary of zone-ids treated as character symbols and uses the dictionary to gather statistics based on movement history contexts, or phrases. Being motivated by the dictionary-based LZ78 compression algorithm [10], our algorithm assumes the name "LeZi-update", pronounced "lazy update" as will be clear in the following.

3.2 The LeZi-update algorithm

The LeZi-update algorithm enhances an underlying update scheme. The algorithm captures the sampled message, or movement history, and processes it in chunks. Updates are encoded and reported periodically as a sequence "$C(w_1)C(w_2)C(w_3)\ldots$", where the w_is are segments of the movement history and $C(w)$ is the encoding for segment w. The prime requirement for LeZi-update (following LZ78) is that the w_is must be distinct.

The system's knowledge about the mobile's location always lags by at most the gap between two updates. The uncertainty increases with this gap, yet larger gaps reduce the

Procedure Encoder

```
loop
  wait for next symbol v
  if(w.v in dictionary)
    w := w.v
  else
    encode <index(w),v>
    add w.v to dictionary
    w := null
forever
```

Procedure Decoder

```
loop
  wait for next codeword <i,s>
  decode phrase := dictionary [i].s
  add phrase to dictionary
  increment frequency for every
                prefix of phrase
forever
```

number of updates. A natural action would be to delay the update if the current string segment being parsed has been seen earlier, i.e., the path traversed since the last update is a familiar one. Although the gap increases, it is expected that the information lag will not affect paging much if the system can make use of the profile generated so far. This prefix-matching technique of parsing is the basis of the LZ78 compression algorithm, which encodes variable length string segments using fixed-length dictionary indices, while the dictionary gets continuously updated as new phrases are seen. We outline this greedy parsing technique as used in our context. The mobile acts as the *encoder*, while the system takes the role of the *decoder*. For Bob's movement history, the Lempel-Ziv parsing boils down to the phrases $m, a, mc, mr, k, d, kd, g, o, og, dk, l, r, mcm, am, rl, kdl, rm, amc, mrk, dkd, go, \ldots$, where the commas represent the points of updates.

The LZ78 algorithm emerged out of a need for finding a universal variable-to-fixed length coding scheme, where the coding process is interlaced with the learning process. The key to the learning is a de-correlating process, which works by efficiently creating and looking up an explicit dictionary. The algorithm in [10] parses the input string "$v_1, v_2, \ldots, v_n$", into $c(n)$ distinct substrings $w_1, w_2, \ldots, w_{c(n)}$ such that for all $j \geq 1$, the prefix of substring w_j (i.e., all but the last character of w_j) is equal to some w_i, for $1 \leq i < j$. Because of this prefix property, substrings parsed so far can be efficiently maintained in a multiway tree or a *trie*.

We can further improve the performance of the algorithm by augmenting the dictionary with the suffixes of decoded phrases on the decoder as shown below. The enhanced trie for our example takes the shape as in Figure 3, where the frequency counts of the phrases are shown within parentheses. The symbol Λ denotes an empty string.

Enhanced Decoder

loop
  wait for next codeword <i,s>
  decode phrase := dictionary [i].s
  add phrase to dictionary
  increment frequency for every prefix of every suffix of phrase
forever

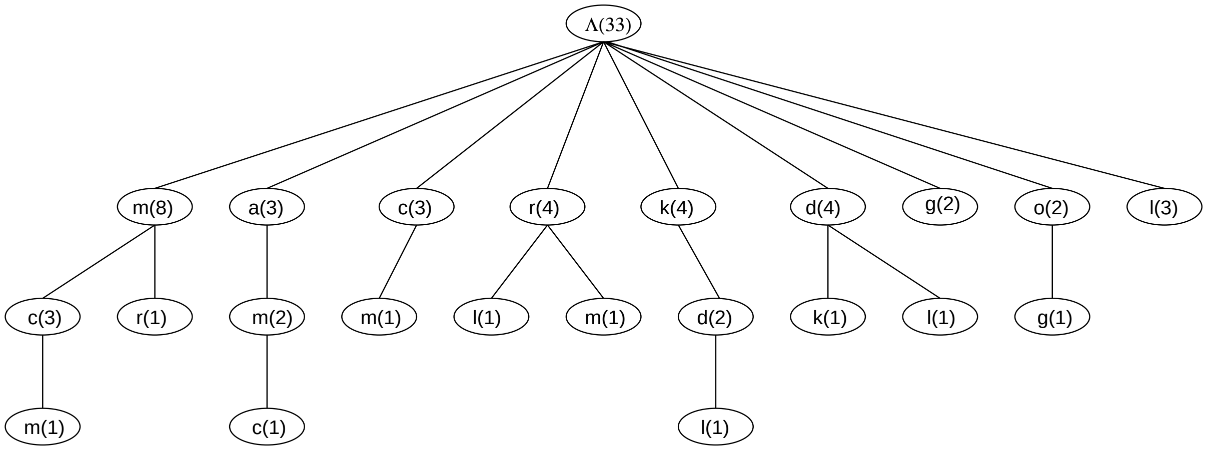

Figure 3: Trie for the enhanced LZ symbol-wise model.

As the process of incremental parsing progresses, increasingly larger phrases accumulate in the dictionary. Consequently, estimates of conditional probabilities for larger contexts start accruing. Intuitively, the process would gather the predictability or richness of higher and higher order Markov models. Since there is a limit to the model richness for stationary processes, the symbol-wise model should eventually converge to the universal model.

Each user's location database holds a trie, which is the symbol-wise context model corresponding to the enhanced Lempel-Ziv incremental parse tree. Each node except for the root preserves the relevant statistics that can be used to compute total probabilities of contexts as well as conditional probabilities of the symbols based on a given context. A path from the root to any node w in the trie represents a context, and the sub-trie rooted at w reveals the conditional probability model given that context. The paths from the root to the leaves in the Lempel-Ziv trie represent the largest contexts which are not contained in any other contexts.

Next, we assign probabilities for the occurrence of the symbols (zones) on the path segment to be reported by the next update. These segments are the sequences of zones generated when traversing from the root to the leaves of the sub-trie representing the current context. The estimated conditional probabilities for all the zones at the current context constitutes the conditional probability distribution.

We use the enhanced trie in Figure 3 to illustrate a blending strategy known as *exclusion*. Assume that Bob's predicted location is needed, no LeZi-style path update message has been received since receiving the last "*amc*" in the sequence. The contexts which can be used are suffixes of "*amc*," with the exception of itself. First, we need to find all possible paths that can be predicted at these contexts. A list of all such paths are shown in Table 2, with their respective frequencies. The unconditional probabilities of occurrence of these phrases are then computed by blending. The phrase "m", for example, appears in the contexts of all the orders 0, 1 and 2. We start from the highest order, i.e., the context "*mc*". The phrase "m" occurs only once out of three possible occurrences of this context, the other two producing null prediction. Thus we can predict the phrase "m"with probability $\frac{1}{3}$ at context "*mc*" and fall back to the lower order with probability $\frac{2}{3}$. Now "m" occurs once at the context "c", out of a total three occurrences of the context. Thus, "m" can be predicted with probability $\frac{1}{3}$ at the order-1 context. Due to two occurrences out of the three producing null predictions, we need to escape to lower order with probability $\frac{2}{3}$. Finally, "m" shows up four times out of thirty-three possible phrases, leading to a probability value $\frac{4}{33}$. The blended probability of phrase "m" is thus $\frac{1}{3} + \frac{2}{3}\{\frac{1}{3} + \frac{2}{3}(\frac{4}{33})\} = 0.6094$. Since the phrase is made of only one symbol 'm', the whole probability is assigned to that symbol.

aa (order-2)	a (order-1)	Λ (order-0)		
$m\|mc(1)$	$m\|c(2)$	$m(4)$	$c(2)$	$kdl(1)$
$\Lambda\|mc(2)$	$\Lambda\|c(2)$	$mc(2)$	$cm(2)$	$d(2)$
		$mcm(1)$	$r(2)$	$dk(1)$
		$mr(1)$	$rl(1)$	$dl(1)$
		$a(1)$	$rm(3)$	$g(2)$
		$am(1)$	$k(2)$	$o(1)$
		$amc(1)$	$kd(1)$	$og(1)$
				$l(3)$

Table 2: Phrases and their frequencies at contexts "aa", "a" and "Λ".

We have shown here that location learning is one prediction task essential for intelligent environments. The LeZi-update algorithm builds user profiles to perform this prediction based on the LZ78 compression algorithm. The effectiveness of this algorithm is apparent for wireless service providers [3], and we are now in the process of demonstrating its use for intelligent environments such as MavHome.

4 Inhabitant Action Prediction

In the MavHome environment, the intelligent agent representing the house needs to predict the inhabitant's next action in order to automate the routine and repetitive tasks for the inhabitant. Patterns observed in past inhabitant activities can be used to aid the agent decisions for controlling devices throughout the home. In this section we describe this second role of prediction in the MavHome architecture and present a sequence matching approach to performing inhabitant action prediction based on collected histories of actions.

Prediction is a heavily researched area in artificial intelligence. The ONISI system [5] and the Korvemaker and Greiner UNIX command prediction algorithm [6] employ pattern matching. IDHYS [8] represents an approach to action prediction based on the Candidate Elimination algorithm.

Our proposed SHIP (Smart Home Inhabitant Prediction) algorithm matches the most recent sequence of events with sequences in collected histories. SHIP considers matches of length three or greater, and returns matches of sufficiently high value based on the *match frequency* (number of occurrences of the pattern in the history) and *match length* (length of the matched sequence).

In the SHIP algorithm, the inhabitant commands are encapsulated using *actions* and *matches*. When the inhabitant issues a command to a device, it is recorded as an action in the inhabitant *history*. A match identifies a sequence in history that matches the immediate event history (a sequence ending with the most recent event). The SHIP's predicted action corresponds to the action that followed the matched sequence most frequently in the inhabitant history. A match queue is maintained to ensure a match time that is close to linear.

The algorithm SHIP consists of two steps. First, the match queue is updated when a new action is recorded. At time t in state s we compute $l_t(s, a)$, the length of the longest sequences that end with action a in state s and match the history sequence immediately prior to time t. In addition, we define a frequency measure $f(s, a)$ which represents the number of times the action a has been taken from the current state. In the second step, the matches in the queue are evaluated based on the match length and frequency.

To allow for gradual changes in the inhabitant patterns over time, the value of a matched pattern can be multiplied by a user-specified decay factor. The user also has the flexibility of weighting the match length and match frequency factors that affect a match value. Because an inhabitant pattern is likely to contain small variations between occurrences, an inexact match is employed to find sequence matches. The user can specify an *inexact threshold* which represents the maximum percentage of mismatches that may occur in a matched sequence.

To test the predictive accuracy of the SHIP algorithm, we equipped devices in several homes with X10 controllers and collected action histories for these households with different numbers of people, types of activities, and spans of time. For more precise experiments we created a synthetic data generator which simulates several possible activity scenarios. With each type of activity is associated a Gaussian probability distribution over start times and durations from which actual event histories can be generated.

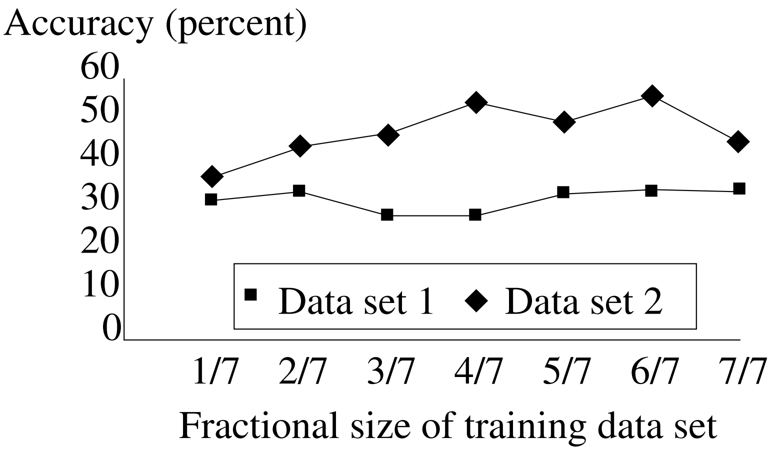

Figure 4: SHIP predictive accuracy results.

The graph in Figure 4 shows the percentage of correct predictions for two of the data sets. For this experiment a decay factor of 0.0 is used and the match frequency and length weights are set to 1.0. These data sets reflect activities collected over a maximum period of 30 days. Data set 1 captures 747 activities of four inhabitants with 11 devices and data set 2 represents 3,000 activities of one inhabitant using 16 devices.

As the figure shows, the accuracy of the algorithm generally improves as the amount of training data, calculated as a percentage of the total history, increases. The greatest predictive accuracy for data set 1 is 33.3% and for data set 2 is 53.4%. The accuracy rate reflects greatly improved performance over a random guess. As a separate test, we measure the predictive accuracy made by one of the top three matches identified by SHIP. In this case, the predictive accuracy is over 80%. These results indicate that the SHIP algorithm is effective for predicting inhabitant actions in a smart home, particularly if the top several matches are considered.

5 Discovery of Significant Patterns

The SHIP algorithm is useful in identifying likely activities of a smart home inhabitant. This information can be used to automate interactions with the home, removing the need for manual control of devices. A wrong prediction, however, can be annoying or detrimental if the inhabitant must undo the action executed by the house or repair damage caused by a faulty decision.

Instead of identifying and automating each inhabitant pattern, we describe here a prediction algorithm, called Episode Discovery (ED), that identifies *significant episodes* within an inhabitant event history. A significant episode can be viewed as a related set of device events that may be ordered, partially ordered, or unordered. A significant episode occurs at some regular interval or in response to other significant episodes called *triggers*. The goal of the intelligence framework in our problem domain is to mine the input stream in order to discover the significant episodes. Actions can then be automated based on the significance of the discovered pattern as well as the predictive accuracy of the next event.

Our approach is based on the work of Srikant and Agrawal [9] for mining sequential patterns from time-ordered transactions. Our home automation problem differs from previous research in that the input sequence does not consist of explicit transactions, but merely interactions with home devices. Unlike the previous sequence mining problem, the significant episodes in an intelligent environment may be ordered (sequential) or unordered (member of a set). In addition, many of the episodes in our environment will occur on a regular basis (daily, weekly) and need to be recognized for this regularity. In our MavHome scenario, the following device activity sequences occur on a regular basis and should be detected by our algorithm:

- HeatOn (daily)
- AlarmOn, AlarmOff, BedroomLightOn, CoffeeMakerOn, BathRoomLightOn, BathRoomVideoOn, ShowerOn, HeatOff (daily)
- BedroomLightOff, BathRoomLightOff, BathRoomVideoOff, ShowerOff, KitchenLightOn, KitchenScreenOn (daily)
- CoffeeMakerOff, KitchenLightOff, KitchenScreenOff (daily)
- HotTubOn (daily)
- HotTubOff (daily)
- SprinklerOn (weekly)

- SprinklerOff (weekly)
- VCROn (weekly)
- VCROff (weekly)
- OrderGroceries (weekly)

Other activities, such as the robot activation, would not be identified by ED as significant because they do not occur with any predictable regularity.

To mine the data, the input sequence is partitioned into transaction-like collections of events by sliding a window over the event history and viewing the collection of events within the window as an unordered set. The minimum description length (MDL) principle is used to evaluate potential sequences. The MDL principle targets patterns that can be used to minimize the description length of the database by replacing each instance of the pattern with a pointer to the pattern definition. This evaluation measure thus identifies patterns that balance frequency with pattern length. As a result, automating these sequences will significantly reduce the amount of necessary interaction between an inhabitant and the environment. Another feature of ED is that patterns are evaluated for day, week, and month regularity as well as MDL value. Significant episodes will then be selected based on the overall evaluation measure, and used as the basis for activity prediction and home automation.

The episode discovery problem is defined as follows. Let E be the set of all device events. An *event occurrence* O is a pair (e, t) relating an event e with an integer time value t. An *event sequence* S is an ordered sequence of event occurrences. We define an *episode* $\mathcal{E}$ as a set of event occurrences, and a *candidate itemset* I as a set of events and episodes, $I = \{e_1, e_2, e_3, e_n\}, \{\mathcal{E}_1, \mathcal{E}_2, \ldots, \mathcal{E}_m\}$, where each event e_i has an occurrence in each of the episodes $\mathcal{E}_j$, for $1 \leq i \leq n$ and $1 \leq j \leq m$. A *significant episode* L is an episode that meets or exceeds the evaluation threshold, and an *event sequence description* D is a description of an event sequence using significant episodes and event occurrences.

1. Construct an event sequence S from input O.
2. Partition the S into episodes using a sliding window of length w.

3. Create candidate itemsets from the episodes.

4. Compute compression values for each of the candidate itemsets.

5. Using a greedy approach, identify the candidate itemset that minimizes the description length of the set of episodes as a significant episode.

6. Remove all of the episodes associated with the candidate itemset from the remaining candidate itemsets.

7. Remove all candidate itemsets that have an empty episode set.

8. Repeat steps 4 - 7 until the list of candidate episodes is empty.

An on-line incremental version of ED has been implemented using C++. To validate this algorithm, a synthetic 28-day data set was generated reflecting our MavHome scenario. To make the problem more challenging, noisy events were included, episodes were overlapped, and the times of the event occurrences were varied each day. Using a fifteen minute time window, ED discovers the eleven significant episodes described by the scenario. Using a twelve hour window, the daily and weekly occurrences are detected as separate significant episodes. These results show that ED can be used to aid in the automation of device interactions, as described by our MavHome scenario.

6 Conclusions

In this paper we have presented the MavHome smart home architecture, which allows a smart home (or other intelligent environment) to act as a rational agent. As a rational agent the home receives input from sensors and selects an appropriate action which is executed through the use of effectors. This architecture allows the integration of research in machine learning, databases, mobile computing, robotics, and multimedia computing that is essential for smart home development.

As part of the MavHome architecture, several prediction algorithms are introduced that play critical roles in an adaptive and automated environment such as MavHome. The first prediction algorithm, the LeZi-update algorithm, provides an optimal approach to the location management problem that is useful in determining the position of an inhabitant for routing messages and multimedia information. This novel approach uses movement histories to learn likely future locations. The second algorithm, smart home inhabitant prediction (SHIP), uses sequence matching with inexact allowances and decay factors to determine the most likely next inhabitant interaction with the home. Results from synthetic and real collected smart home data indicate that the predictive accuracy is high even in the presence of many possible activities. The final algorithm, episode discovery (ED), uses the principle of minimum description length to determine which episodes in an inhabitant history are significant. Significance is determined based on the ability to compress the description length of the history as well as periodicity. As a result, these episodes represent events that should be automated by MavHome.

We have demonstrated the effectiveness of these algorithms on collected data. The next step for this effort will be to implement the architecture in the context of actual smart environments. In these contexts we will show the effectiveness of the MavHome architecture operating as a rational agent, and its ability to improve the lifestyle of inhabitants in a variety of intelligent environments.

References

[1] I. F. Akyildiz and J. S. M. Ho. Dynamic mobile user location update for wireless pcs networks. *Wireless Networks*, 1(2):187–196, 1995.

[2] I. F. Akyildiz and J. S. M. Ho. Movement-based location update and selective paging for pcs networks. *IEEE/ACM Transactions on Networking*, 4(4):629–638, 1995.

[3] A. Bhattacharya and S. K. Das. Lezi-update: An information-theoretic approach to track mobile users in pcs networks. In *Proceedings of the Annual International Conference on Mobile Computing and Networking*, pages 1–12, 1999.

[4] Y. Birk and Y. Nachman. Using direction and elapsed-time information to reduce the wireless cost of locating mobile units in cellular networks. *Wireless Networks*, 1(4):403–412, 1995.

[5] P. Gorniak and D. Poole. Predicting future user actions by observing unmodified applications. In *National Conference on Artificial Intelligence*, 2000.

[6] B. Korvemaker and R. Greiner. Predicting UNIX command lines: Adjusting to user patterns. In *Proceedings of the Conference on Intelligent Applications of Artificial Intelligence*, 2000.

[7] G. P. Pollini and C.-L. I. A profile-based location strategy and its performance. *IEEE Journal on Selected Areas in Communications*, 15(8), 1997.

[8] J.-D. Ruvini and C. Dony. Ape : Learning users habits to automate repetitive tasks. In *International ACM Conference on Intelligent User Interfaces*, pages 229–232, 2000.

[9] R. Srikant and R. Agrawal. Mining sequential patterns: Generalizations and performance improvements. In *Proceedings of the Fifth International Conference on Extending Database Technology (EDBT)*, 1996.

[10] J. Ziv and A. Lempel. Compression of individual sequences via variable-rate coding. *IEEE Transactions on Information Theory*, 24(5):530–536, 1978.