Using Apriori algo to find frequent repeated patterns and their periodicity.

Keeping the Resident in the Loop: Adapting the Smart Home to the User

Parisa Rashidi, *Member, IEEE,* Diane J. Cook, *Fellow, IEEE,*

***Abstract*—Recent advancements in supporting fields have increased the likelihood that smart home technologies will become part of our everyday environments. However, many of these technologies are brittle and do not adapt to the user's explicit or implicit wishes. Here we introduce CASAS, an adaptive smart home system that utilizes machine learning techniques to discover patterns in resident's daily activities and to generate automation polices that mimic these patterns. Our approach does not make any assumptions about the activity structure or other underlying model parameters, but leaves it completely to our algorithms to discover the smart home resident's patterns. Another important aspect of CASAS is that it can adapt to the changes in discovered patterns based on the resident implicit and explicit feedback and can automatically update its model to reflect the changes. In this paper we provide a description of the CASAS technologies and the results of experiments performed on both synthetic and real world data.**

***Index Terms*—Smart environments, Machine learning, User-centered design, Adaptive systems.**

I. Introduction

RECENTLY there has been extensive research toward developing smart environments by integrating various machine learning and artificial intelligence techniques into home environments that are equipped with sensors and actuators. A smart environment acquires and applies knowledge about the physical setting and its residents, in a way that can optimize a number of different goals including maximizing comfort of the residents, minimizing the consumption of resources, and maintaining safety of the environment and its residents [1]–[4]. Application of smart environment technologies has encompassed different areas such as interactive conference rooms, offices, kiosks, and furniture with seamless integration between heterogeneous devices that facilitate collaborative work environments [5]. To achieve intelligent behaviors in smart home applications, various computational intelligence techniques have been proposed to support the creation of smart homes, including neural networks [6], fuzzy logic [7], hierarchical task network planning [8], hidden Markov Models [9], and Bayesian networks [10]. Due to the difficulty of creating an automated physical environment, many smart environments ideas are discussed in theory or are tested just on synthetic data. In those cases where physical environments have been designed [1], [3], [4], [6], [7], the culmination of the project is an environment with sensing and automation. In none of these projects is the focus placed on creating an environment where the resident can guide the environment to behave in a customized manner.

Smart environments have the potential to aid people with cognitive and physical limitations, to provide resource conservation, and to make our lives more comfortable and productive. For example, by discovering repetitive sequences, modeling their temporal constraints and learning their expected utilities, we can intelligently automate repetitive daily tasks in homes. The long-term goal of many smart home projects is to automate resident interactions with the environment that are repetitive or, in the case of individuals with physical limitations, are difficult to perform.

A primary hindrance to realizing the potential benefits of smart homes is the ease with which smart environment technology can be integrated into the lifestyle of its residents. Our goal is to design a smart environment that finds repetitive patterns in resident's activities, and adapts to changes in those patterns. Considering the fact that humans usually change their habits and activities over time, adaptation is a crucial part of a smart environment solution. However despite its importance, most smart home researchers [1], [3], [4] assume that the learned model of resident behavior is static over the lifetime of the system. One of the few works done in this area is a primitive approach proposed by Valtonen et al. [11], where a fuzzy controller adjusts the weights of a few predefined factors that determine whether performing a specific action is in accordance with the user's preferences or not. However, they do not consider the problem of adapting to more complex sequential activities, and do not discuss how such changes can be discovered from daily activity data. In addition, they consider a set of predefined factors, but in the real world, it is not possible to determine such factors in advance. In our work, we consider automation of sequential complex activities that adapts to the user's preference. As already mentioned, our approach does not make any assumptions about the activity structure or other underlying model parameters. In addition, we will utilize context information such as temporal information or startup triggers that helps to better model the complex environment. With our approach, the resident plays a critical role in guiding the environment's automation policy. Specifically, the resident can guide the system by providing explicit feedback, or s/he can leave it to the system to automatically discover and adapt to changes in pattern of activities.

II. 2. CASAS Components

Our smart home system, CASAS, contains several cooperating components that find activity patterns, generate policies

P. Rashidi is with the Department of Electrical Engineering and Computer Science, Washington State University, Pullman, WA, 99163 USA, e-mail:prashidi@eecs.wsu.edu.

D. J. Cook is with the Department of Electrical Engineering and Computer Science,Washington State University, Pullman, WA, 99163 USA e-mail: cook@eecs.wsu.edu.

Manuscript received October 20, 2008; revised February 14, 2009.

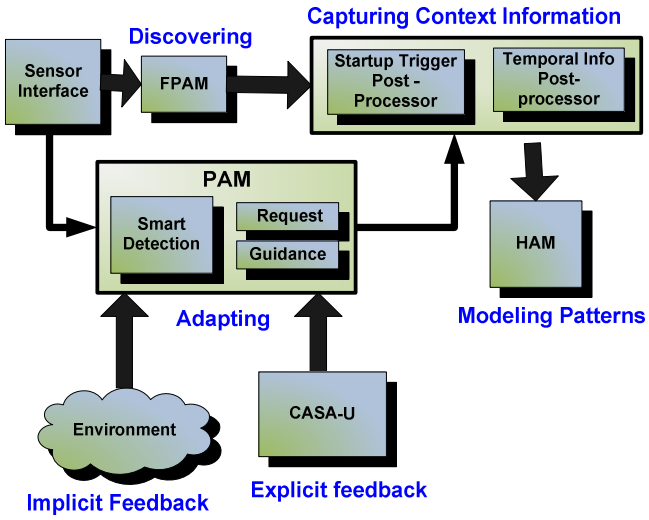

Fig. 1: CASAS software architecture.

to automate these patterns, predict and schedule automated activities, and adapt to explicit user feedback or observed changes in resident behavior. Fig. 1 depicts the interaction between these components.

Input to CASAS consists of sensor data collected by the smart environment, such as motion sensors and light sensors. This data is mined by our FPAM algorithm to discover activity patterns of interest for automation. These patterns are then modeled by our Hierarchal Activity Model (HAM) to further utilize the underlying temporal and structural information as part of related context. The Pattern Adaptation Miner (PAM) algorithm adapts to any changes in those patterns and responds to user guidance. The adaptation is based on using four mechanisms: direct manipulation (using our CASA-U user interface), guidance, request, and smart detection. Each method requires a certain balance between user collaboration and system autonomy, from no user involvement in the smart detection method to no autonomy in the direct manipulation method. Putting all the adaptation methods together, we are able to provide an adaptive smart environment solution which adapts to its residents over time, and which provides a wide range of collaboration and feedback methods such that residents can choose to act proactively or passively to customize the smart environment. In the rest of the paper, we will describe each component in more detail.

III. Frequent and Periodic Activity Miner: FPAM

The first step of automating activities in a smart home is discovering patterns of the resident's activity from sensor data collected daily. Data collected in a smart home consists of readings by various sensors (e.g., motion, light, temperature) and resident manual manipulation of lights and appliances, as captured by power-line controllers. Discovering how the resident performs routine activities in daily life facilitates home automation and makes it possible to customize the automation for each person. In this work we primarily consider discovering frequent and periodic activities, as automating these activities makes the environment responsive to the resident and removes the burden of repetitive tasks from the user. In our discussions, we define an event as a single action such as turning on the light, while an activity is a sequence of such events; for example the "turning on the light - turning on the coffee maker" activity is composed of two events. We will use the terms pattern, sequence, and activity interchangeably, depending on the context. In this work, we restrict our discussion to single-resident environments. While others have dealt with the problem of multiple residents [12], in our approach data collected on multiple residents would be modeled as a single, more complex, entity.

To find frequent and periodic patterns of activity, we will be exploiting a data mining subfield usually referred to as "discovery of frequent sequences" [13], [14], "sequence mining" [15], or "activity monitoring" [16]. In this area, the pioneering work of Agrawal's Aprori algorithm [15] was the starting point, later extended by other researchers [17]. We use a variant of the Apriori algorithm to find frequent patterns. However, in smart environment applications, not only it is important to find frequent activities, but also those that are the most regular, occurring at predictable periods (e.g., weekly, monthly). If we ignore periodicity and only rely on frequency to discover patterns, we might discard many periodic events such as the sprinklers that go off every other morning. Therefore we need a unified framework that can simultaneously detect both frequent and periodic activity patterns. There are a number of earlier works that try to handle periodicity, such as the Episode Discovery (ED) algorithm [4]; however, the problem with ED and most other approaches is that they look for patterns with an exact periodicity, which is in contrast with the erratic nature of most events in the real world. Lee, et al. [18] define a period confidence for patterns, but they require the user to specify a set of desired pattern time periods. In our approach, we do not require users to specify predefined periods, as it is not realistic to force users to know in advance which time periods will be appropriate. In addition, the periodicity of patterns for many practical applications is dynamic and thus changes over time. To overcome this problem, we define two different periodicity temporal granules, to allow for the necessary flexibility in periodicity variances over two different levels. A fine grained granule is defined for hourly periods which can span several hours up to 24 hours, and a coarse grained granule is defined for daily periods which can span any arbitrary number of days. None of these periodicity granules require a period to be exact: fine grained periods provide a tolerance of up to one hour and coarse grained periods provide a tolerance of up to one day.

In our model, we capture important context information such as startup triggers as well as temporal information including event durations and start times. Startup triggers are events that can trigger another action, e.g. a person entering a room can act as a potential startup trigger for the light in the room to be turned on. Triggers, which are absent in most traditional data mining methods, are a key contextual aspect in smart environments that need to be processed accordingly. But despite their importance, previous mining methods applied to smart home applications have ignored the startup trigger concept, and they typically do not differentiate between real sensor data (startup triggers) and actuator data (data obtained

TABLE I: Sample of collected data.

source (d_i)	state (v_i)	Timestamp (t_i)
$Light_1$	ON	05/15/2007 12:00:00
$Light_2$	OFF	05/15/2007 12:02:00
$Motion_4$	ON	05/15/2007 12:03:00

from appliances through power-line controllers). In addition, most techniques treat events in the sequence as instantaneous and ignore the conveyed temporal information. Laxman and Sastry [13] do model some temporal information, by incorporating event duration constraints into the episode description. Similarly, Lee et al. [18], associate each item in a transaction with a duration time. Bettini et al. [19] place particular emphasis on the support of temporal constraints for multiple time granularities where the mining process is modeled as a pattern matching process performed by a timed finite automaton. Our work is different from these previous approaches, as we mostly focus on estimating time distributions for different time granules by utilizing combinations of local Gaussian distributions for modeling temporal information of each pattern, rather than merely considering temporal granules. Using a combination of multiple Gaussian and several temporal granules allows us to more accurately express and model duration and start times. In the following sections, we will describe the FPAM algorithm and its mechanism in more detail.

A. Finding Candidate Patterns

We assume that the input data is a sequence of tuples that appear in the form $\langle d_i, v_i, t_i \rangle$, where d_i denotes a single data source (e.g., motion sensor, light sensor, appliance), v_i denotes the state of the source (e.g., off, on), and t_i denotes the time that the event occurred. We assume that data does not arrive in stream format. Instead, it is sent to a storage media, and the data mining process is carried out offline at regular time intervals such as weekly, or on demand as will be described later. Table. I shows a sample of collected data. Our FPAM algorithm, similar to the Apriori method, takes a bottom-up approach. However, unlike the Apriori algorithm, not only does it discover frequent sequences, but it also tries to find periodic sequences and their periodicity.

In the first iteration, a window of size ω (initialized to 2) is passed over the data and every sequence of length equal to the window size is recorded together with its frequency and initial periodicity. Frequency is computed as the number of times the sequence occurs in the dataset, and periodicity represents the regularity of occurrence, such as every three hours or weekly. After the initial frequent and periodic sequences have been identified, FPAM incrementally builds candidates of larger size. FPAM extends sequences that made the cutoff in the previous iteration by the two events that occur before and after the current sequence instances in the data. For simplicity, we call the event right before the current sequence the sequence's prefix and the event right after it the sequence's suffix. FPAM incrementally increases the window size until no frequent or periodic sequences within the new window size are found or a user-defined limit on the window size is reached. For example,

Fig. 2: Frequent pattern "BC".

Fig. 3: Extending "BC" pattern by its prefix.

Fig. 4: Extending "BC" pattern by its suffix.

Fig. 5: Extending "BC" pattern by both its suffix and prefix.

consider Fig. 2, where "BC" is a frequent pattern. Now if it is extended by its prefix (Fig. 3), then it results in another frequent pattern "ABC". However, extending it by its suffix (Fig. 4) or by both suffix and prefix (Fig. 5) does not result in a frequent pattern. Each discovered sequence is considered as being either periodic or frequent. At the end of the mining session, if a specific sequence is found to be both frequent and periodic, for convenience and simplicity we report it as frequent.

B. Evaluating Candidate Patterns

Drawing on results from information theory, we evaluate the frequency of a sequence based on its ability to compress the dataset by replacing occurrences of the pattern by pointers to the pattern definition. Calculating this amount of compression is tricky for smart home data, as the size of the dataset may not be fixed due to varying activity levels (e.g. a particularly active day will generate a lengthy event dataset). We compute compression according to (1), where f_a represents the frequency of sequence a, t represents the input data length in hours and C represents the compression threshold. For a sequence to be considered as frequent, the following condition should hold:

$$\frac{|a| * f_a}{t} > C \tag{1}$$

Previous approaches [19] have calculated the input data length in numbers of tuples, rather than in units of time, which results in making discoveries of frequent patterns dependent on the resident's activity level. For example, if the resident has a very active day, the input data will contain more tuples and therefore the input length will take on a high value, but if resident is not active that day, the input length will have a low value. Therefore for an activity with a constant frequency such as making coffee twice a day, the compression value will be dependent on the resident's activity level. In our approach, the input length is measured in time units rather than tuples, and an activity such as making coffee twice a day will be discovered as a frequent pattern, independent of the activity level of the resident. In addition to finding frequent patterns,

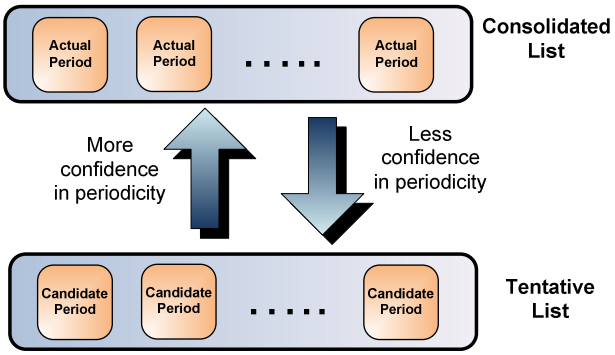

Fig. 6: Moving periods between consolidated and tentative lists.

FPAM also discovers periodic patterns. Calculating periods is a more complicated process. To calculate the period, every time a sequence is encountered, we will compute the elapsed time since its last occurrence. More precisely, if we denote the current and previous occurrence of a sequence pattern as s_c and s_p, and their corresponding timestamps as $t(s_c)$ and $t(s_p)$, then the distance between them is defined as $t(s_c) - t(s_p)$. This distance is an initial approximation of a candidate period. To determine periodicity, as mentioned before, two different periodicity granules are considered: coarse grained and fine grained period granules. A fine grained granule is defined for hourly periods which can span several hours up to 24 hours, and a coarse grained granule is defined for daily periods which can span any arbitrary number of days. To construct periods, a lazy clustering method is used, such that as long as an activity's period can be matched with the previous periods (with a tolerance of one hour for fine grained, and one day for coarse grained), no new period is constructed. If the new activity has a period different than previous periods, a new period is constructed and is added to the tentative list of fine grained or coarse grained periods. In order to make sure that candidate periods are not just some transient accidental pattern, they are kept in a tentative list until they reach a confidence frequency value. If the periodicity for a sequence is consistent a threshold number of times (e.g. 90%), the pattern is reported as periodic, and it is moved to the consolidated period list. Updating tentative and consolidated lists is performed dynamically and a period can be moved from one list to another several times (see Fig. 6). Such a schema helps to eliminate any transient periods based on current or future evidence, resulting in an adaptive evolvable mining approach. In this approach, whenever more data becomes available as a result of regularly scheduled mining sessions, the periods are revisited again, and if there is any period that does not meet periodicity criteria anymore, it will be moved from the consolidated list into the tentative list. Later if we find again more evidence that this period can be consolidated, it will be moved back into the consolidated list; this results in a more robust model that can evolve and adapt over time.

As we scan through the data, we calculate and update the expected number of occurrences, $E(f_a)$, for an activity a, up to the current point in the data. For example, if our initial period estimate for activity a is 5 hours and so far we have scanned through 10 hours of data, we expect to see two occurrences of activity a in an ideal case. Considering the erratic nature of real-world events, not all sequences will be repeated ideally. To deal with this problem, for each new occurrence of a we check it against the following equation where f_a is actual number of occurrences observed so far and ζ is a predefined threshold that determines what percentage of expected occurrences is sufficient to move a candidate periodic sequence from the tentative list into a consolidated list (e.g. a rigorous approach can consider it to be above 95%). If (2) holds for a pattern a, we will consider it as a periodic pattern.

$$\frac{E(f_a)}{f_a} > \zeta \tag{2}$$

The discovered patterns will form the basis of CASAS' automated activities. To facilitate effective automation FPAM records the duration and start times of events within each pattern by processing their corresponding timestamps. A distribution of these values is generated from individual values and passed to HAM in order to generate an automation model from the mined sequence patterns.

C. Triggers

An important notion that can be used to improve activity prediction in smart environments is the discovery of sequence startup triggers. Basically, a trigger is an event which causes an activity to start. A startup trigger paradigm can be compared to the event programming paradigm, in which for example a mouse click event (startup trigger) can trigger an action (such as a menu appearing). In smart environments, the same paradigm applies; for example, if a person enters a dark room, it can be considered as a startup trigger for turning on the light; or as another example, running out of milk in the refrigerator can be a trigger to initiate a supermarket purchase reminder. These startup triggers will also appear in the collected data and therefore it is necessary to augment the data mining model with a trigger processing component that is able to recognize triggers, in order to facilitate automation of activities.

A trigger is typically part of an FPAM's discovered sequence. For example, if a person turns on the light every time s/he enters the room, FPAM will discover the sequence "enter room - turn on light" from the sensor and power-line controller data. By examining this sequence, we will find out that a startup trigger is not actually part of an automated activity; rather it is a condition that starts an automated activity (in this case, turning on the light). We also can see that the startup triggers consist of sensor events, while automated events are generated by actuators (power-line controllers attached to appliances). In our model, we will process the discovered sequences from FPAM in such a way that a sequence merely represents automations and only contains data from actuators, though it can have several startup triggers assigned to it. For example, the previous sequence "enter room - turn on light" will be converted to a single event sequence "turn on light" with the "enter room" triggers assigned to it. A sequence can have several triggers assigned to it. We adopt the following general policy for processing triggers:

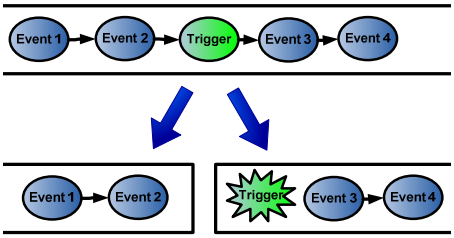

Fig. 7: A trigger causing a sequence to be split.

- If a trigger appears at the beginning of a sequence, it should be removed from the beginning of the sequence and be added to the list of assigned start up triggers.
- If a trigger appears at the end of a sequence, it has no effect on the sequence; we simply remove it from the sequence.
- If several triggers occur consecutively, we will just consider the last one, discarding the other ones.
- If a trigger occurs in the middle of a sequence, we will split the sequence into two new sequences and the trigger will be assigned to the second sequence (see Fig. 7).
- If a sequence contains more than one trigger, the above steps are repeated recursively.

Note that we assume that frequency and periodicity would be the same for split sequences as the original sequence; but, the compression value may change as it depends on the sequence's length. So, the compression value is computed for recently split sequences and if it does not satisfy the frequency criteria, recently split sequences will be removed from the frequent patterns' list. Also during the sequence splitting process, sequences might reduce to one of the already existent sequences. In this case, one approach is to repeat the data mining process again to find any existing relation between these two sequences (e.g., they might have different periods). However, for the sake of simplicity and also efficiency, we do not mine the data again; rather we will choose the sequence with the highest compression value and simply discard the other.

IV. 4. Hierarchical Activity Model: HAM

After activity structures and periods have been discovered by FPAM, the sequences will be organized in a Hierarchal Activity Model (HAM) structure, which filters out activities according to two temporal granule levels of day and hour (see Fig. 8). In addition to finding frequent and periodic patterns, FPAM records durations and start times of events by processing their timestamps. These durations and start times are revisited by PAM when looking for changes. HAM captures the temporal relationships between events in an activity by explicitly representing sequence orders in a tree structure containing Markov chains at the bottom (or sensor) level [20]. Each activity will be placed in a HAM leaf node (sensor level) according to its day and time of occurrence, and will have a start time and event durations assigned to it. Earlier approaches to modeling durations of states in a Markov chain such as the approach by Vaseghi [21] condition state transition probabilities on how long the current state has been occupied. In our model, for each activity at each time node, we describe the start time and the duration of each individual event using a normal distribution that will be updated every time new data is collected (see Error! Reference source not found.). If an activity is located in different time/day nodes within the HAM model, multiple Gaussian distributions for that activity will be computed, allowing HAM to more accurately approximate the start time and duration values, by using multiple simple local normal functions that represent a more complex function. CASAS automatically constructs and updates the HAM model from FPAM data and uses the model to identify activities that need to be automated at a given time. The HAM model can be then used to find activities that should be automated at a given time. HAM provides greater support for representing and learning automation policies than earlier approaches that model all observed activities in one level of a highly-connected single Markov model or simple hierarchies just based on location [4].

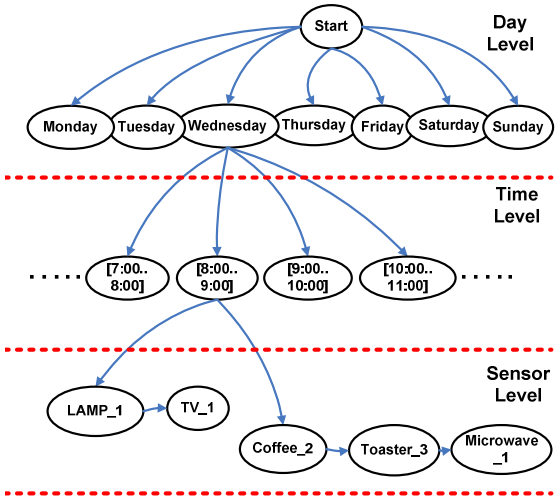

Fig. 8: Example HAM structure.

A. Selecting Actions for Automation

After the hierarchical model is constructed, HAM selects activities to be automated. Such automation should reflect a preference-based adaptation policy, such that recently discovered activities will not miss their chance of being selected for automation simply because not enough time has not passed to allow them be explored by the learning algorithm. To achieve this, we require the following conditions to hold: 1) the highest-likelihood activities are given a greater chance of being automated, 2) less likely activities retain a chance of being automated (especially recently discovered ones) and 3) the temporal relationships between activities are preserved. CASAS is not currently designed to handle the intricacies associated with interleaving multiple automation activities that overlap in time. As a result, not all actions that appear in the hierarchy for a particular time can necessarily be automated.

HAM selects activities for automation that maximize the expected utility [20] as shown:

$$EU(A) = P_T(A) * Q(A) \quad (3)$$

In (3), for a given activity A, $Q(A)$ is A's potential value which reflects the amount of evidence against or for an activity as being frequent or periodic (will be described more later), and $P_T(A)$ is the total occurrence probability of activity A, defined as:

$$P_T(A) = P_d(A) * P_t(A) * P_r(A) \tag{4}$$

In (4), $P_d(A)$, the daily probability, reflects the occurrence probability of A on a given day of the week. The time probability, $P_t(A)$, reflects the occurrence probability of A in a given time interval (node); and the relative probability, $P_r(A)$, reflects the relative occurrence probability, with respect to the other activities that fall within the same time node. The daily and time probabilities are estimated using frequencies computed by FPAM and the relative probability is computed using P_{d_i} and P_{t_i} for all activities that fall within the same time node.

To select an activity, CASAS balances exploration and exploitation where exploration of potential automated activities allows for potential improvement of the smart home, and exploitation avoids user frustration due to too many wrong choices. The probability of selecting a particular activity A for automation is calculated according to (5). Here, $EU(j)$ represents the expected utility of activity j as defined in (3), $\beta * D(A)$ is a term that favors recently-added activities, $D(A)$ represents how recently activity A occurred (as a reciprocal of the number of days since its discovery); β adjusts the relative importance of recently-added activities vs. highly expected activities; and k is a parameter which initially is set high to promote exploration, but over time decreases, to allow for exploitation of stabilized automations.

$$P(A) = \frac{k^{EU(A)+\beta*D(A)}}{\sum_j k^{EU(j)+\beta*D(j)}} \tag{5}$$

V. Dynamic Adaptation

Learning a model of resident's activities provides a basis for automating activities in a smart home. However, this is not a long-term solution because residents are likely to change their activity patterns over time depending on factors such as changing job or social relations, seasonal and weather conditions, and mental and emotional condition. As a result, we need to find a way to adapt to the changes that occur over time, in addition to incorporating explicit resident guidance of automation policies.

CASAS achieves adaptation based on the resident's explicit feedback, provided through the CASAS user interface, or based on implicit feedback which can be described as any alteration in the resident's habits and lifestyle. For example, consider a resident that used to turn on the coffee maker every day at 7:30 am, but later changes his habit and turns it on at 6:30 am. This is an example of implicit user feedback that should be detected by CASAS. We employ four different adaptation mechanisms to consider the resident's explicit and implicit feedback: direct manipulation, guidance, request and smart detection.

A. Updating the Model

As already mentioned, CASAS provides four different methods for adaptation. In direct manipulation, residents provide the system with the most explicit form of feedback, by manipulating automated activities using the CASAS user interface, CASA-U. Using the guidance method, residents guide CASAS by rating the automated activities in CASA-U based on their preferences on scale of $1..5$, thus providing CASAS with explicit feedback or advice. With the request method, residents can highlight any activity to be monitored by CASAS for possible changes, therefore providing a mixture of explicit and implicit preference feedback. In the last approach, called smart detection, CASAS utilizes the most implicit form of feedback by monitoring resident activities and updating the HAM model. The difference between the last two methods is first how fast the changes will be detected; and second the required amount of user interaction, as smart detection method doesn't require any user interaction. Using the request method, the change detection process starts immediately and hence residents do not have to wait for the regular mining schedule. CASAS uses all of these mechanisms to provide a flexible, user-centric solution to the dynamic adaptation problem to allow for various degrees of resident involvement. Residents can choose any of above methods to provide feedback to CASAS, and can choose to act proactively or be passive.

For every activity, we maintain a potential value, Q, which reflects the amount of evidence against or for an activity as being frequent or periodic, in other words the degree to which it should be considered for automation. The potential value can be increased or decreased through a compensation effect or a decay effect, as will be described. If potential value falls below a certain activity threshold, the activity is discarded from the model, in other words it will be forgotten. Maintaining a potential value for each discovered activity can help us distinguish transient changes from long-term changes that still might be accompanied by a noise element. The potential value is increased by using the following formula:

$$Q = Q + \alpha * r \tag{6}$$

In (6), $r \in [-1...+1]$ denotes the evidence value, and α denotes the learning rate. To simulate the overriding nature of learning in the direct manipulation and guidance methods, we set the learning rate to a relatively high value; while for the request and smart detection methods we set it to a small value to simulate their gradual history-preserving nature. Note that when updating the potential value, we do not differentiate between different events that comprise an activity and consider it as a whole; therefore we assign a single value to an activity.

In addition to the compensation effect, we also employ a decay effect which subtracts a small value ϵ (decay rate) from all activities' values at each time step θ. Applying decay function, the value of any activity during an arbitrary time interval Δt is decreased by:

$$Q = Q - \frac{\epsilon * \Delta t}{\theta} \tag{7}$$

The decay effect allows for those activity patterns that have not been perceived over a long period of time to descend toward a vanishing value over time, or in an intuitive sense to be forgotten. This helps CASAS to adapt to the changing preferences of residents. The effect of the decay function is kept in check through a compensation effect because the potential value remains bounded.

B. Detecting changes

In request method, whenever a pattern is highlighted to be monitored, PAM analyzes recent event data and looks for changes in the pattern, such as the pattern's start time, durations, periods, or the pattern structure (the component events with their temporal relationships). Without loss of generality, we refer to two different categories of changes: changes that preserve the structure and changes that alter the structure. Structure change is detected by finding new patterns that occur during the same times we expect the old pattern to occur; assuming that the start time can act as a discriminative attribute. First, PAM looks for a pattern, a, such that its start time, s_a, is contained within the interval $\Delta\delta = \mu_a \pm \sigma_a$, where μ_a and σ_a denote the mean and standard deviation of the original pattern's start time distribution. These locations are marked by the algorithm in order to avoid looking at all data. PAM is looking for different patterns within these start time intervals, in which we expect to see the original pattern. It moves a sliding window of size ω (initially set to 2) over the interval and incrementally increases the window size after every iteration. The window size does not increase when no more frequent or periodic patterns of length ω can be found. A frequent pattern can easily be extended beyond the marked point, as we require only its start time to be contained within the marked interval. This process results in finding a new pattern which may be longer, shorter, or have other different properties than the original pattern.

In the case where structure is preserved, we first mark all the occurrences of the original activity in the data, and based on these occurrences calculate properties such as new durations, new start times or new periods. After results from both cases have been collected, PAM reports the list of changes that can be accepted or rejected by the user.

In the smart detection method, PAM automatically mines the data regularly to update the model and uses the decay and compensation effects to adapt to changes. This approach is slower than the explicit request method, because the changes might not be detected until the next scheduled mining session. After every mining session, the discovered patterns will include a mixture of new and previously-discovered patterns. For new patterns, we simply can add them to the model. For previously existing patterns, if the pattern shows no change, then PAM applies the compensation effect to indicate observation of more evidence for this pattern (6). However, if the pattern shows some changes, we will add the modified patterns to the model, while also preserving the original pattern, as there is no explicit evidence that this change is a permanent change. To achieve adaptation in this case, we will leave it to the compensation effect and decay functions to decide over time which version is more likely. The compensation effect will increase the value of the more frequently-observed version of the pattern while the decay function dominates for a less-observed pattern. As a result, the value of patterns that have not been observed for a long time will fall below the activity threshold; and will eventually be removed from the model. This again results in adapting to the changes in the environment. The minimum mining period that results in convergence of patterns can be ascertained theoretically [22].

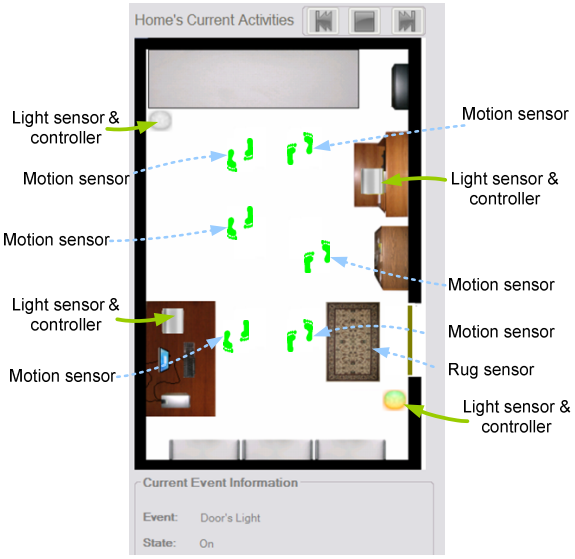

Fig. 9: 2D visualized model of AI lab and corresponding sensor layout.

VI. CASA-U: The User Interface

Several issues make design of smart home interfaces challenging. One such challenge is the choice of representation for smart home activities. Our objective is to present the smart home and its automation policies to the user in a clear manner, using a floor plan of the home as a primary means of communicating this information. We also need to represent the spatial relationship between elements in the home, the status of these elements, and the temporal nature of associated actions.

We consider our smart environment interface, CASA-U, as a discrete event simulator where each object is a self-descriptive, iconic representation of an item in the environment. For preliminary studies, the sensor layout and floor plan of the visualizer was based on the sensor layout and floor plan of the AI lab at Washington State University as a simple smart environment test-bed. The portion of the lab that we use for our experiments is modeled as shown in Fig. 9, which is equipped with motion sensors, rug sensors, light sensors, and Insteon controllers for the lamps. To show the effect of motion sensors detecting someone walking around the room, we use footprints to imply the motion effect.

CASA-U allows the resident to control the events that are distributed across time as well as the resident's living space. To achieve this, CASU-U creates a temporal framework and spatial framework to allow the resident to perceive, comprehend, and ultimately modify events occurring in the physical world

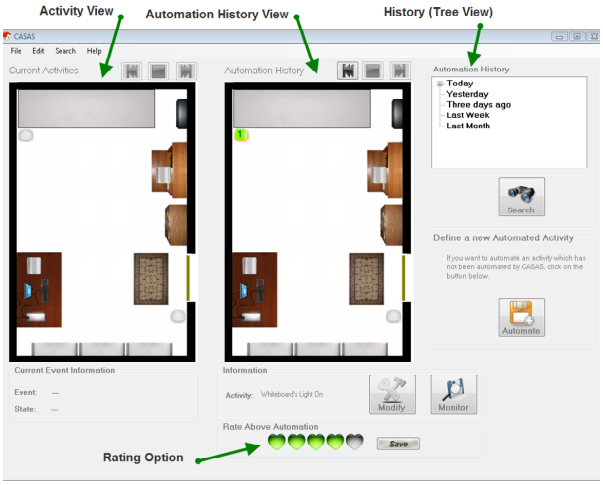

Fig. 10: Main view of CASA-U.

around the resident. In our schema, the floor map provides a spatial framework and the temporal constraints are displayed as an animation of event sequences where the direct mapping of the order of events in the physical world maps to the order of the displayed elements. In order to provide a clearer view of the temporal relations that exist between the automation events, we label the objects with numbers that indicate their temporal ordering. Two spatial views are available to the user (see Fig. 10): one visualizes a live stream of sensed events (activity view) and the other animates past automated CASAS activities (automation history view). In both views, the user has the option to go backward or forward in the stream using rewind/forward buttons to find a specific event.

A challenge we faced in the design of the CASA-U user interface was how to provide users with the ability to change scheduled automated activities or request new automations. Each activity's model includes a definition of constituent events, the relative and absolute temporal relationships between these events, the duration of each event, startup triggers and periods. Given this complexity, it is essential to provide the user with adequate guidance. In our design, whenever a user wants to define or modify an activity, s/he is guided through a series of wizard dialogs where each dialog asks the appropriate question based on previous steps and in each step, a brief description about that step is provided in order to help users better understand the underlying conceptual model. Activity modification in CASA-U uses a direction manipulation approach. Users can click on an item in the map and change its attributes using either a context menu or drag and drop. For example, to define a new activity that includes the sequence "turn on the desk lamp then turn on the whiteboard light", the user may first right click on the desk lamp, select "turn on" from the context menu and repeat the step for the white board light, or alternatively drag them into the activity panel. Users can define or change any aspect of activities, such as temporal information or startup triggers. Fig. 11 shows the first step of a wizard dialog where user is being asked for the types of changes s/he would like to make. Fig. 12 shows its next step where user selects the events that

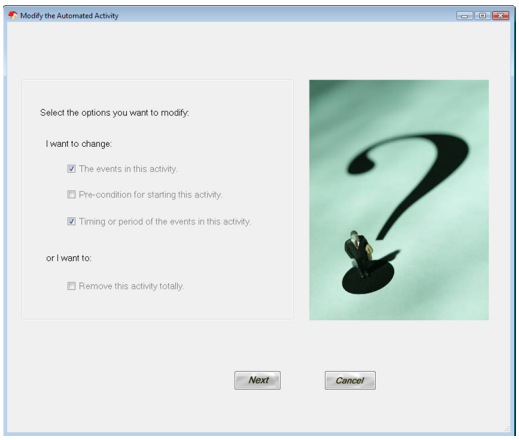

Fig. 11: Wizard dialogs guide user through modifications.

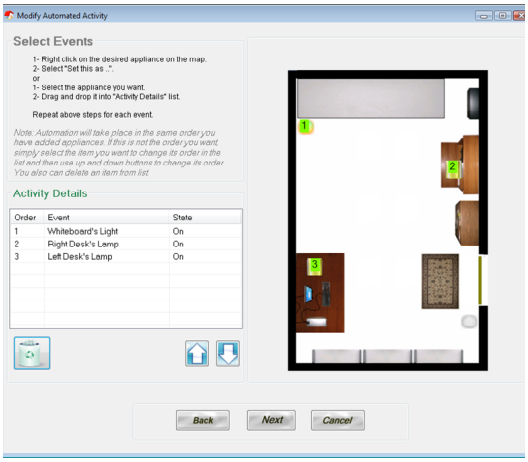

Fig. 12: Modifying an automation by selecting events to be modified.

need to be modified. The numbers in Figure 12 indicate in what order the automation should be performed. In addition, the user can modify the duration of each event, as shown in Fig. 13. There is also an option for scheduling automations, such that the automation is applied any number of times on the scheduled dates (e.g. on holidays), or repeatedly based on regular periods (e.g. hourly, daily, weekly). Users also have the option to search through the previous history, using a tree view element (see Fig. 10), using the forward/backward playback capability of automation history view, or alternatively restricting the history view by identifying a number of search criteria and viewing the list of matched results in an animated way.

CASA-U also provides an option for users to highlight an activity to be monitored (corresponding to request method). Once changes have been detected in the highlighted activity, a dialog is popped up to show the user a summary of the detected change. The user then has the option to apply some of all of the detected changes to the model for future automation, or to reject the changes.

Another feature of CASAS is its ability to adapt its automation policies based on explicit feedback from the resident. Explicit feedback includes direct manipulation of the model

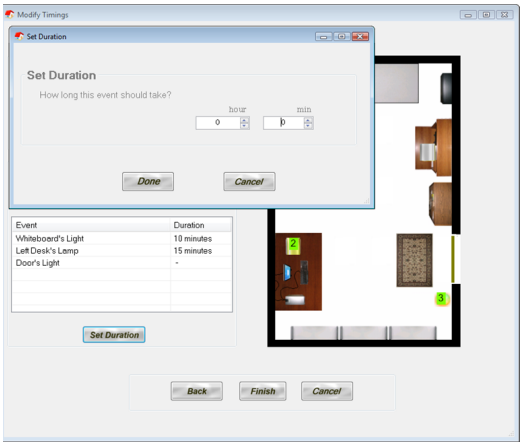

Fig. 13: Modifying an automation by setting duration of each event.

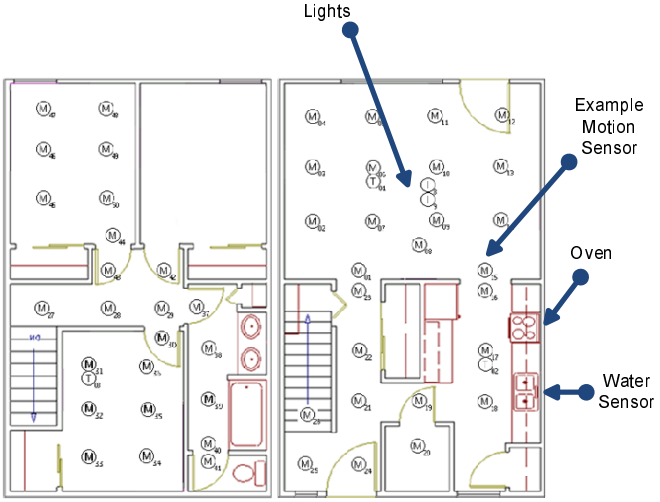

Fig. 14: The layout of sensors in smart apartment.

or ratings (guidance) that indicate the degree to which the resident likes (or does not like) an automation policy. For each policy, the user can provide a rating between 1 and 5. PAM then updates its HAM model automatically based on this user guidance by changing the activity's potential value according to (6), where the ratings will be mapped to corresponding evidence values.

The usability studies performed on CASA-U [23] revealed that the interface was relatively clear and easy to navigate, but additional training is needed for residents to make effective use of smart home technologies.

VII. Experiment Results

The goal of this project was to design smart home system that could automatically learn policies that reflect resident behavior and that could effectively incorporate implicit and explicit feedback from the user into the models. In this section we evaluate the ability of the CASAS software to meet this goal using synthetic-generated data as well as and real data collected from a residential apartment as well as our lab-based physical testbed.

A. Evaluation of FPAM

We first want to evaluate FPAM's ability to find frequent and periodic patterns from smart home event data. To test the algorithm on synthetic data, we implemented a synthetic data generator that simulates events corresponding to a set of specified activities. Timings for the activities can be varied and a specified percentage of random events are interjected to give the data realism. In addition to synthetic data, to evaluate FPAM on real world data, we tested it on data obtained through sensors located in one room of the CASAS smart workplace environment. In addition, we performed evaluation on data that was collected in the CASAS smart apartment..

For the first experiment, we generated one month of synthetic data that represented a number of different frequent and periodic activity patterns in a home. The data contained various random activities stretched over one month, along with several target activities that either appear frequently, or repeat based on regular periods. To test the effect of startup triggers, we included several triggers such as "sitting on couch", "walking to TV", "walking to living room" and "sitting on bed". The periods for these activities ranged from 6 to 24 hours. Details of these target activities are listed in Table. II.

Our expectation was that FPAM would correct identify all target activities among random events with their corresponding periods, startup triggers, start times, and durations. In fact, FPAM was able to find all of the activities with their correct periods. In addition, FPAM identified triggers where they existed. When faced with two events that are scheduled to occur at the same time, the synthetic data generator randomly picks just one of the events to simulate. Because of this design feature, the frequency of some of the detected activities was not as high as expected and thus the compression rates were lower than anticipated.

In addition to these synthetic data experiments, we also tested FPAM on real smart environment data collected in a smart apartment equipped with motion sensors, light sensors, water sensors, and some appliance controllers (Fig. 14). The data contained 62 hours of events that span four days of daily activities for two residents living in the apartment and contains approximately 19,000 recorded events. Again we ran FPAM on data to see how well it can find patterns in large dataset. FPAM was again able to discover some patterns in data, for example "M15: ON - Oven: High - Water: Running", which shows a pattern where the resident walks up to turn on the oven, and then opens the water tap. In this case, M15, a motion sensor (all motion sensors start with M in layout), acts as a startup trigger. Another example is pattern "M17: ON- Oven: High-Oven: Low", which shows that the resident walks up to change the oven from high to low. The periodic patterns were far less common in this dataset; probably due to the fact the participants were students with somehow chaotic schedules, therefore most patterns were frequent with no particular periodicity.

Because only a few appliances are currently equipped with controllers the number of emergent patterns is small. However,

TABLE II: Target activities (”-” indicates the activity is frequent as opposed to being periodic).

	Start time	Period	Events
1	6:00	6 hours	(1) Kitchen-Light-ON (2) Coffee-maker-ON (3) Toaster-ON
2	-	-	(1) Sitting-on-Couch (2) Couch-Light-ON (3) Music-ON
3	7:00	24 hours	(1) Bathroom-light-ON (2) Water-Hot
4	10:00	24 hours	(1) Windows-Blind-OFF
5	15:35	12 hours	(1) Sprinklers-ON (2) Walking-to-living-room
6	-	-	(1) Walking-to-TV,TV-ON
7	19:00	6 hours	(1) Dining-Table-light-ON (2) Music-ON (3) Coffee-maker-ON
8	-	-	(1) Sitting-on-Bed (2) Lamp-ON

the results indicate how FPAM will work on real large datasets.

We also tested FPAM on data obtained in our AI lab testbed. We recruited a participant to execute a simple script in the smart testbed environment. The participant moved through the environment shown in Fig. 9 for about an hour, repeating the script ten times. In order to inject some randomness into the data, the participant was asked to perform random activities for about one minute in step three of the script. The script is defined as follows:

1) Turn on right desk light, wait 1 minute.
2) Turn off right desk light.
3) Perform random activities for 1 minute

Because the testbed area was fairly small, the participant inadvertently created patterns in the random actions themselves. In addition, the motion sensors picked up slight motions (such as hand movements) which resulted in a randomly-triggered pattern that occurred between steps 1 and 2 in the script, which FPAM then split into two subsequences. Despite these issues that occur in a real-world situation, FPAM was able to accurately discover the following patterns:

- Right desk lamp on, Compression: 12, Trigger: Walking nearby
- Right desk lamp off, Compression: 9
- Left desk lamp on, Compression: 2
- Right desk lamp on, Right desk lamp off, Compression: 10
- Whiteboard light on, Compression: 2

The first and second patterns are the result of splitting the sequence ”Right desk lamp off, Random event, Right desk lamp on” into two subsequences. The ”walking” trigger is correct because after turning the light off, the participant performs a random action and heads back to the right desk to turn on the light, which usually involves walking across the room to reach the desk. The difference in compression values between the first and second sequences is due to multiple triggers from the light sensor for a single light on or light off action. The third sequence is the result of a random activity; the compression value is relatively small compared to the main script activities. The fourth sequence reflects the embedded activity, and the last sequence is a frequent activity associated with random events, again with a smaller compression value. These results support our claim that FPAM can detect patterns correctly.

In another experiment, in order to test the effect of noisy data on the sequence discovery procedure, we injected different percentages of noise into the data. We used the same data set as in the first experiment (refer to Table. II). For this experiment, we set the compression value to 0.3, and the fine and coarse grained periodic confidence to 0.9. To inject noise, a certain percentage of sequences in data were changed randomly, in a way that the whole sequence structure was different with respect to the original sequence (every single event in the sequence was changed).

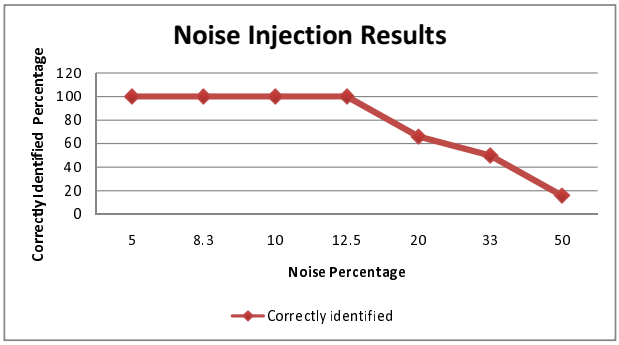

Fig. 15: Noise injection results.

As it can be seen in Fig. 15, FPAM acts robustly despite the presence of noisy data up to 12.5% noise (100% of the original sequences are correctly identified), but after that point, injecting more and more noise into the system leads to less and less regularity in patterns such that eventually the sequence’s compression value will fall below the compression threshold (in this case, 0.3). It is possible to lower the compression threshold in order to increase robustness. The resulting algorithm might find less frequent sequences too, which may not be very interesting for us.

In another experiment, in order to test the effect of noisy data on the sequence discovery procedure, we injected different percentages of noise into the data. We used the same data set as in the first experiment (refer to Table. II). For this experiment, we set the compression value to 0.3, and the fine and coarse grained periodic confidence to 0.9. To inject noise, a certain percentage of sequences in data were changed randomly, in a way that the whole sequence structure was different with respect to the original sequence (every single event in the sequence was changed).

As it can be seen in Fig. 15, FPAM acts robustly despite the presence of noisy data up to 12.5% noise (100% of the original sequences are correctly identified), but after that point,

injecting more and more noise into the system leads to less and less regularity in patterns such that eventually the sequence's compression value will fall below the compression threshold (in this case, 0.3). It is possible to lower the compression threshold in order to increase robustness. The resulting algorithm might find less frequent sequences too, which may not be very interesting for us.

B. Evaluation of PAM

In order to evaluate PAM's ability to adapt to new patterns, we again tested it on both synthetic and real data. We hypothesize that PAM can adapt to changes in discovered patterns. To test the hypothesis, for our first experiment we created one month of synthetic data with six embedded scenarios, the same as in previous experiment with FPAM. After FPAM found corresponding activity patterns, we highlighted the third activity to be monitored for changes. We then changed the activity description in the data generator such that all event durations were set to 8 minutes, instead of 5 minutes. PAM detected the changes accordingly by finding a new duration of 7.3 minutes, which is quite close to the actual 8 minute change. The data generator does have an element of randomness, which accounts for the discrepancy between the specified and detected time change. In similar tests, PAM was also able to detect start time changes from 19:00 to 19:30, and structure changes (omission or addition).

We next tested our adaptor on real world data using the AI lab. A volunteer participant entered the room and executed two different scripts:

1) Turn on right lamp (1 min), perform random actions
2) Turn on left lamp (1 min), perform random actions

The first activity was repeated 10 times over the course of two hours with random events in between. Then the participant highlighted the activity for monitoring and performed the second scripted version by changing the duration from 1 to 2 minutes. PAM detected the duration change as 1.66 minutes. The change was made to the correct parameter and in the correct direction, but did not converge on an accurate new value due to the detection of other similar patterns with different durations. These experiments validate that PAM can successfully adapt to resident changes even in real-world data. We also found that in addition to changes in duration, PAM detected some changes in start time. This is another correct finding by PAM. As in the second dataset, we changed the duration of all events in all scenarios which resulted in a shifted start time for all scenarios, in our case 14:55 instead of original 14:25.

We also empirically validated our theoretical analysis [22] to see how fast original patterns will be replaced by modified versions, by generating two sets of synthetic data and validating the adaptation capability for different decay values (see Fig. 16). Our findings are consistent with our expectation, validating that PAM can successfully adapt to resident changes.

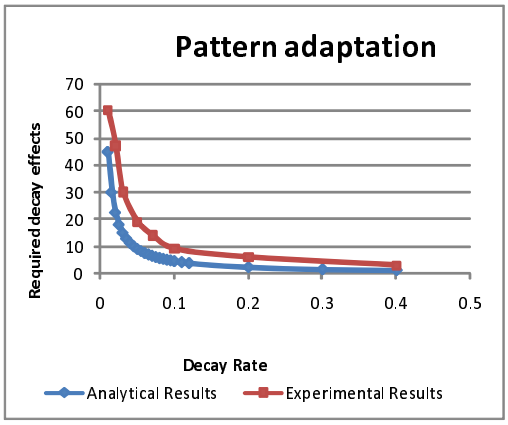

Fig. 16: Number of decay effects required to forget a pattern, with respect to decay rate.

VIII. Conclusions

In this paper, we presented CASAS, an integrated set of components that aim toward applying machine learning and data mining techniques to a smart home environment in order to detect activity patterns, generate automation policies for those patterns, and also adapt to the changes in those patterns. In our ongoing work, we plan to perform more user studies in real world setting to better understand the strengths and weaknesses of the system. We are also planning to extend CASA-U for a real residential apartment, using more realistic 3D modeling. Ultimately, we anticipate adding additional features such as a voice recognition capability to the system to increase availability and ease of use. We also intend to discover additional types of contextual information that allow the model to better generalize over discovered sequences.

Parisa Rashidi Parisa is currently a Ph.D student at Washington State University. She received her B.S.c in computer engineering from University of Tehran, Iran in 2005. In 2007, she received her M.Sc in computer science from Washington State University where she worked on CASAS project as her master thesis. Her interests include smart environments, AI and machine learning applications in health care, and human factors in pervasive computing applications.

Diane J. Cook Dr. Diane J. Cook is a Huie-Rogers Chair Professor in the School of Electrical Engineering and Computer Science at Washington State University. She received a B.S. degree in Math/Computer Science from Wheaton College in 1985, a M.S. degree in Computer Science from the University of Illinois in 1987, and a Ph.D. degree in Computer Science from the University of Illinois in 1990. Her research interests include artificial intelligence, machine learning, and smart environments.

REFERENCES

[1] G. D. Abowd, E. D. Mynatt, and T. Rodden, "The human experience," *IEEE Pervasive Computing*, vol. 1, no. 1, pp. 48–57, 2002.

[2] D. Cook and S. Das, *Smart Environments: Technology, Protocols and Applications (Wiley Series on Parallel and Distributed Computing)*. Wiley-Interscience, 2004.

[3] S. Helal, W. Mann, H. El-Zabadani, J. King, Y. Kaddoura, and E. Jansen, "The gator tech smart house: a programmable pervasive space," *Computer*, vol. 38, no. 3, pp. 50–60, March 2005.

[4] G. Youngblood and D. Cook, "Data mining for hierarchical model creation," *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, vol. 37, no. 4, pp. 561–572, July 2007.

[5] A. Fox, B. Johanson, P. Hanrahan, and T. Winograd, "Integrating information appliances into an interactive workspace," *IEEE Computer Graphics and Applications*, vol. 20, no. 3, pp. 54–65, 2000.

[6] M. C. Mozer, "The neural network house: An environment hat adapts to its inhabitants," AAAI, Tech. Rep. SS-98-02, 1998.

[7] F. Doctor, H. Hagras, and V. Callaghan, "A fuzzy embedded agent-based approach for realizing ambient intelligence in intelligent inhabited environments," *Systems, Man and Cybernetics, Part A, IEEE Transactions on*, vol. 35, no. 1, pp. 55–65, Jan. 2005.

[8] F. Amigoni, N. Gatti, C. Pinciroli, and M. Roveri, "What planner for ambient intelligence applications?" *Systems, Man and Cybernetics, Part A, IEEE Transactions on*, vol. 35, no. 1, pp. 7–21, Jan. 2005.

[9] L. Liao, D. Fox, and H. Kautz, "Location-based activity recognition using relational markov networks," *In Advances in Neural Information Processing Systems (NIPS)*, 2005.

[10] E. M. Tapia, S. S. Intille, and K. Larson, "Activity recognition in the home using simple and ubiquitous sensors," 2004, pp. 158–175.

[11] M. Valtonen, A. M. Vainio, and J. Vanhala, "Continuous-time fuzzy control and learning methods," in *Communications and Information Technologies, 2007. ISCIT '07. International Symposium on*, 2007, pp. 346–351.

[12] A. Crandall and D. Cook, "Attributing events to individuals in multi-inhabitant environments," *Intelligent Environments, 2008 IET 4th International Conference on*, pp. 1–8, July 2008.

[13] S. Laxman and P. S. Sastry, "A survey of temporal data mining," in *Academy Proceedings in Engineering Sciences*, vol. 31. The Indian Academy of Sciences, 2006, pp. 173–198.

[14] J. Roddick and M. Spiliopoulou, "A survey of temporal knowledge discovery paradigms and methods," *Knowledge and Data Engineering, IEEE Transactions on*, vol. 14, no. 4, pp. 750–767, July-Aug. 2002.

[15] R. Agrawal and R. Srikant, "Mining sequential patterns," 1995, pp. 3–14.

[16] T. Fawcett and F. Provost, "Activity monitoring: Noticing interesting changes in behavior," in *In Proceedings of the Fifth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 1999, pp. 53–62.

[17] H. Mannila and H. Toivonen, "Discovering generalized episodes using minimal occurrences." AAAI Press, 1996, pp. 146–151.

[18] C.-H. Lee, M.-S. Chen, and C.-R. Lin, "Progressive partition miner: An efficient algorithm for mining general temporal association rules," *IEEE Transactions on Knowledge and Data Engineering*, vol. 15, no. 4, pp. 1004–1017, 2003.

[19] C. Bettini, X. Sean Wang, S. Jajodia, and J.-L. Lin, "Discovering frequent event patterns with multiple granularities in time sequences," *IEEE Trans. on Knowl. and Data Eng.*, vol. 10, no. 2, pp. 222–237, 1998.

[20] R. Sutton and A. Barto, *Reinforcement Learning: An Introduction*. MIT Press, 1998.

[21] S. V. Vaseghi, "State duration modelling in hidden markov models," *Signal Process.*, vol. 41, no. 1, pp. 31–41, 1995.

[22] P. Rashidi and D. Cook, "Adapting to resident preferences in smart environments," in *Proceedings of the AAAI Workshop on Advances in Preference Handling*. AAAI, 2008, pp. 78–84.

[23] ——, "Keeping the intelligent environment resident in the loop," in *Proceedings of the International Conference on Intelligent Environments*, 2008, pp. 1–9.