

An Adaptive Sensor Mining Framework for Pervasive Computing Applications

Parisa Rashidi , Diane J. Cook

Washington State University, Pullman,
WA, 99164, USA
{prashidi, cook}@eecs.wsu.edu

Abstract. Analyzing sensor data in pervasive computing applications brings unique challenges to the KDD community. The challenge is heightened when the underlying data source is dynamic and the patterns change. We introduce a new adaptive mining framework that detects patterns in sensor data, and more importantly, adapts to the changes in the underlying model. In our framework, the frequent and periodic patterns of data are first discovered by the Frequent and Periodic Pattern Miner (FPPM) algorithm; and then any changes in the discovered patterns over the lifetime of the system are discovered by the Pattern Adaptation Miner (PAM) algorithm, in order to adapt to the changing environment. This framework also captures vital context information present in pervasive computing applications, such as the startup triggers and temporal information. In this paper, we present a description of our mining framework and validate the approach using data collected in the CASAS smart home testbed.

Keywords: Sensor data mining, Sequential mining, Pervasive computing applications, Smart environments, Adaptation.

1 Introduction

With remarkable recent progress in computing power, networking equipment, sensors, and various data mining methods, we are steadily moving towards ubiquitous and pervasive computing, into a world entangled with abundant sensors and actuators. As a result, there is a wealth of sensor data that can be analyzed with the goal of identifying interesting patterns. For example, by discovering repetitive sequences (frequent or periodic), modeling their temporal constraints and learning their expected utilities, we can intelligently automate a variety of tasks such as repetitive daily tasks in homes, or assembly sequences in manufacturing floors. Mining sequences of sensor events brings unique challenges to the KDD community, and the challenge is even heightened more when the underlying data source is dynamic and the patterns change.

In this work, we introduce a new adaptive mining framework for use with pervasive computing applications, which will detect and adapt to changing patterns in the sensor data. These patterns can be expressed as a set of time ordered sequences,

and discovering and adapting to the changes in such sequences can be achieved by using a sequence mining algorithm tailored to the special domain requirements of pervasive computing applications. The pervasive computing special requirements include utilizing context information such as startup triggers and temporal information, a unified framework for discovering periodic and frequent patterns, and most importantly adaptation over the lifetime of the system. 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 data aspect in pervasive computing environments and need to be processed accordingly. At the same time, as already mentioned, discovering and adapting to the changing patterns over the lifetime of the system is a fundamental part of most pervasive computing applications. In this work, we introduce a framework to address these requirements. Providing such an adaptive framework is a significant advantage over previous sequential mining algorithms applied to pervasive computing applications, which assume the learned model is static over the lifetime of the system [1, 6, 16]. In addition, utilizing context information such as startup triggers helps to better model the complex environment.

In our framework, the frequent and periodic patterns are first discovered by the Frequent and Periodic Pattern Miner (FPPM) algorithm which discovers patterns of arbitrary length and inexact periods. The core component of our model, the Pattern Adaptation Miner (PAM) algorithm, detects changes in the discovered patterns over the lifetime of the system and adapts to the dynamic environment. PAM is able to detect changes autonomously or initiate the detection process by receiving feedback from the user. The startup triggers for discovered patterns are detected and updated accordingly during discovery and adaptation process. Also, the proposed mining framework is able to capture and utilize temporal information by modeling the discovered patterns in a hierarchical model called Hierarchical Activity Model (HAM).

In order to show a tangible application of our model, we evaluate it in the context of a popular pervasive computing application: smart environments. In recent years, smart environments have been a topic of interest for many researchers who have the goal of automating resident activities in order to achieve greater comfort, productivity, and energy efficiency [1, 6, 16]. A smart home uses networked sensors and controllers to try to make residents' lives more comfortable by acquiring and applying knowledge about the residents and their physical surroundings. 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, e.g. turning on the light – turning on the coffee maker, which is composed of two events. We also use the terms pattern, sequence, and activity interchangeably, depending on the context. The events can represent data generated by different sensors such as motion sensors, or by a device that is manipulated by a power-line controller, such as home appliances. 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 the next sections, we will describe our model in more detail. We also show the results of our experiments where we validate our approach on data generated by a

synthetic data generator as well as on real data collected in a smart workplace testbed located on campus at Washington State University.

2 Related Works

Despite increasing progress in pervasive computing applications and especially smart environments, the crucial issue of mining sensor data in a changing environment and maintaining an adaptive model of the environment still has not been explored in depth by the research community. An adaptive solution is especially important for smart environments, as humans often change their habits and lifestyle over time. In the smart home research community, very few adaptive solutions have been proposed, such as the simple reactive fuzzy method proposed in [15], which does not consider any complex sequential tasks, and also does not discuss how such changes can be discovered from daily activity data. In data mining community, some adaptive mining techniques have been proposed for data streams, such as the adaptive associative rule mining algorithm [9]; or methods for adapting to the memory, time or data stream rate constraints [5]. The constraint adaptive methods [5] do not consider adapting to the change in the data; in additions, mentioned works [5, 9] do not address the adaptation problem for sequence mining methods where many different aspects of a sequence can be changed over time. We try to address the adaptation problem in smart environments by using the PAM algorithm to discover any changes in the resident's activities data.

Another significant sensor mining requirement in some pervasive computing applications such as smart environments is detecting the contextual information that describes features of the discovered activities. Context information is valuable in order to better understand the activities and potentially automate them. In our model we will capture important context information such as startup triggers as well as temporal information including event durations, and start times. Previous sensor mining methods applied to pervasive computing applications do not deal with startup triggers' concept systematically, and they typically do not differentiate between real sensor data (startup triggers) and actuator data (data obtained from appliances through power-line controllers). As we will see, this concept plays an important role in pervasive computing applications. In addition, other related techniques treat events in the sequence as instantaneous and ignore the conveyed temporal information. Laxman and Sastry [7] do model some temporal information, by incorporating event duration constraints into the episode description. A similar idea is proposed by Lee et al. [8], where each item in a transaction is associated with a duration time. Bettini et al. [3] place particular emphasis on the support of temporal constraints on 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.

The approach we adopt with our model is to first find patterns of interest. To find frequent activities, we exploit a subset of the temporal knowledge discovery field, that usually is referred to as “discovery of frequent sequences” [12], “sequence mining” [2] or “activity monitoring” [4]. In this area, the pioneering work of Agrawal to design the Apriori algorithm [2] was the starting point. Since then, there have been a number of extensions and variations on the Apriori algorithm [10]. We use a variant of the Apriori algorithm to find frequent patterns. However, in pervasive computing applications, such as smart environments, 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 or the weekly house cleaning. 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 [16]; 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. [8] define a period confidence for the pattern, but they require the user to specify either one or a set of desired pattern time periods. In our approach, we do not require users to specify predefined period confidence for each of the 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 change over time as internal goals or external factors change. 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. A more detailed description of the framework is provided in the next section of the paper.

3 Model Description

We assume that the input data is a sequence of individual event tuples. Each tuple is in the form $\langle d_i, v_i, t_i \rangle$ where d_i denotes a single data source like a motion sensor, light sensor, or appliance; v_i denotes the state of the source such as on or off; and t_i denotes the occurrence time for this particular event. We assume that data is not in stream format, rather 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 1 shows a sample of collected data.

Table 1 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_Sensor_1	ON	05/15/2007 12:03:00

Our FPPM 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, FPPM incrementally builds candidates of larger size. FPPM 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 as its prefix, and the event right after it as its suffix. FPPM continues to increase the window size until no frequent or periodic sequences within the new window size are found or a limit on the window size is reached (if declared by user). 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.

The FPPM algorithm is used to discover the initial patterns. In order to detect the changes in the discovered patterns, PAM provides two options. First, a fast demand-based option called *explicit request* allows users to highlight a number of activities to be monitored for changes. Second, a slower automatic option called *smart detection* automatically looks for potential changes in all patterns, based on a regular mining schedule. The explicit request mechanism detects changes in specified patterns, such that whenever a pattern is highlighted for monitoring, PAM collects data and tries to find potentially-changed versions of the specific pattern. These changes may consist of new activity start times, durations, startup triggers, periods, or structure changes (the sequence's events and their order). Structure change is detected by finding new patterns that occur during the times that we expect the old pattern to occur. Other parameters changes are detected by finding the same structure with other different parameters (e.g., different timing or startup triggers). All changes above a given threshold will be considered as different versions of the pattern and will be shown to the user through our user interface. In addition, our smart detection mechanism automatically mines collected data at periodic intervals (e.g., every three weeks) to update the activity model. The smart detection method adapts slower than the explicit

request method; however it removes the burden of specifying activities to be monitored. Figure 1 shows the overall architecture of the framework.

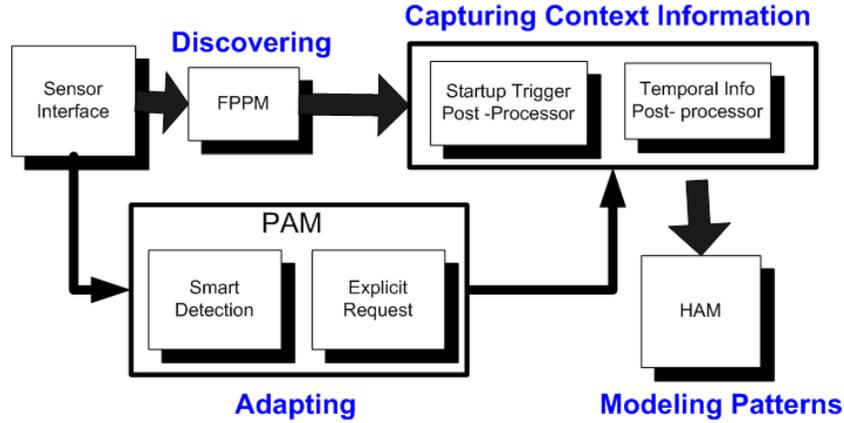


Figure 1 Components of our data mining framework.

3.1 Basic Frequent Pattern Finding

The algorithm finds frequent patterns by visiting the data iteratively. In the first pass, the whole data is visited to calculate initial frequencies and periods, and in the next passes only a small portion of data is revisited. In the first pass, a window ω of size 2, is slid through the entire input data and every sequence of length ω , along with its frequency, f , is recorded. The frequency of each sequence refers to the number of times the sequence is encountered in the data. In order to mark a sequence as a frequent sequence, its frequency should satisfy certain conditions.

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 with pointers to the pattern definition [11]. Calculating this compression is tricky for smart home data, as the size of the dataset may not be fixed due to varying activity levels (e.g. an active day will generate a lengthy event data stream). We compute the compression according to Equation 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:

$$\frac{|a| * f_a}{t} > C \quad (1)$$

Previous approaches [3] 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 the same 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.

3.2 Basic Periodic Pattern Finding

In addition to finding frequent patterns, FPPM 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. 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. One can claim that only a fine grained period can be sufficient to show periodicity of an activity; for example, every Sunday can be represented by a period of 7×24 hours instead of a coarse period of 7 days. This claim is not substantiated in practice, as taking such an approach will require the activity to happen every 7×24 hours with a tolerance of just 1 hour. This is not a realistic assumption, as we want to allow for more tolerance in coarse grained periods. For example, consider the scenario when a resident might watch TV every Sunday, but at different times; in this case, a fine grained period is not able to catch periodicity as its tolerance is just one hour, while a coarse grained period is easily able to catch such a periodicity as it allows for a tolerance of up to one day. The same claim can be made about other time granules, but for sake of simplicity and demonstrating the basic idea, we will just consider the two levels of temporal granules.

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 (see Figure 2). Updating tentative and consolidated lists is performed dynamically and a period can be moved from one list to another several times. 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.

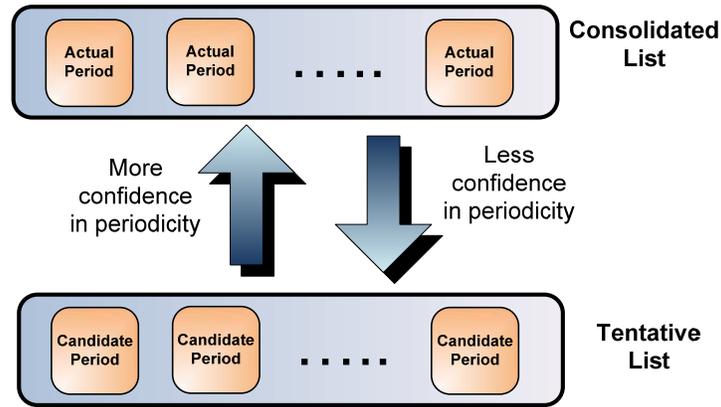


Figure 2 Moving periods between consolidated and tentative lists.

In contrast to frequent pattern finding which uses a single frequency (compression) threshold for detecting all frequent sequences, we cannot use a single confidence threshold for detecting all periodic sequences. Rather we need to tailor the confidence threshold to each specific periodic pattern, because the number of times that an activity occurs can vary depending on its period, making a low frequency sequence a valid periodic candidate in some cases, and a relatively high frequency sequence an invalid candidate in other cases. For example, consider a scenario where our input file contains two weeks of resident activity data and there are two periodic activities: a_1 with a period of one hour and a_2 with a period of 4 days. In this scenario, the number of times we expect to see a_1 would be much more than a_2 . Therefore, a single confidence value cannot work for both cases. To work around this problem, as we scan through the data we calculate and update the expected number of occurrences, $E(f_a)$, for an activity a , up to current point in 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 tentative list into consolidated list (e.g. a rigorous approach can consider it to be above 95%).

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

The predefined threshold can be different for coarse grained and fine grained periods, which we denote as ζ_f and ζ_c . In the next section, we will describe in more detail how the iterative model constructs sequences of larger sizes from the basic sequences.

3.3 Iterative Pattern Discovery

After the initial frequent and periodic sequences of length two have been identified in the first iteration, the next iteration begins. In repetitive iterations, FPPM does not revisit all the data again. Instead, the algorithm attempts to extend the window by the two events before and after the discovered frequent or periodic sequence instances (as we mentioned before, the prefix and suffix) and determines if the extended sequences again satisfy the periodicity or frequency criteria. If we denote the length of the sequence before extending as l , then the extension process might result in finding a frequent or periodic sequence of length $l+1$, if it has been extended by either prefix or suffix; and it might result in finding a frequent or periodic sequence of length $l+2$, if it has been extended by both its prefix and suffix. For example, consider Figure 3, where "BC" is a frequent pattern. Now if it is extended by its prefix (Figure 4), then it results in another frequent pattern "ABC". However, extending it by its suffix (Figure 5) or by both suffix and prefix (Figure 6) does not result in a frequent pattern.

ABC E F A BC G H A BC W Z A BC D X

Figure 3 Frequent pattern "BC".

ABC E F A BC G H A BC W Z A BC D X

Figure 4 Extending "BC" pattern by its prefix.

A BC E F A BC G H A BC W Z A BC D X

Figure 5 Extending "BC" pattern by its suffix.

ABC E F A BC G H A BC W Z A BC D X

Figure 6 Extending "BC" pattern by both suffix and prefix.

Incrementing the window size will be repeated until no more frequent (periodic) sequences within the new window size are found, or a limit on the window size is reached (if declared by the user). The symmetric expansion of window allows for

patterns to grow both forward and backward in time, and the incremental expansion allows for discovery of variable-length patterns. To provide more efficient access to the sequences, at first iteration all sequences of length 2 are stored in a primary hash table and then later discovered sequences (frequent or period) are stored in a multi-level hash table where each level contains sequences of the same size (see Figure 7).

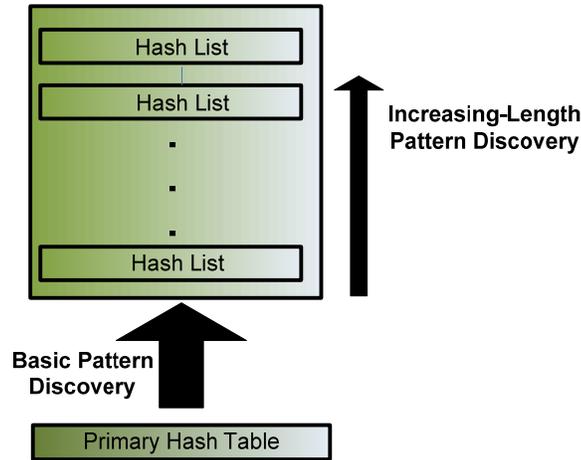


Figure 7 Multi-level hash table to store patterns.

3.4 Startup Trigger Information

An important notion that can be used to improve activity prediction in real world data mining applications such as smart environments is the discovery of startup triggers in sequences. 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 pervasive computing and 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 FPPM's discovered sequence. For example, if a person turns on the light every time s/he enters the room, FPPM 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 light). We also can see that the startup triggers are in fact the sensor data, while automated activities represent data from actuators (power-line controllers attached to appliances). In our model, we will process the discovered

sequences from FPPM 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:

- 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 sequences and the trigger will be assigned to the second sequence (see Figure 8).
- If a sequence contains more than one trigger, the above steps are repeated recursively.

Note that we assume that frequency and period 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, sequence 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.

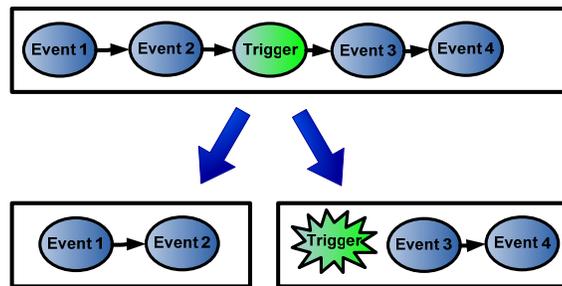


Figure 8 A trigger causing a sequence to be split.

3.5 Startup Trigger Information

After activity structure and periods have been discovered by FPPM, the sequences will be organized in a Hierarchical Activity Model (HAM) structure, which filters out activities according to two temporal granule levels of day and hour (see Figure 9). In addition to finding frequent and periodic patterns, FPPM records duration 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 ordering as a Markov chain [13]. 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.

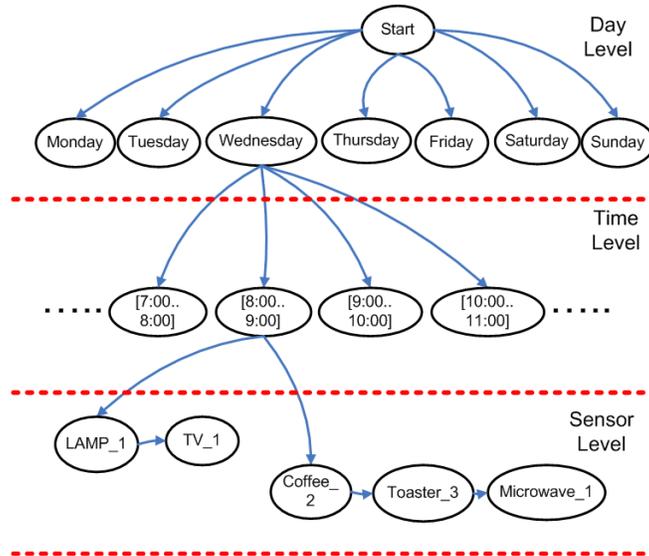


Figure 9 Example HAM model.

There have been earlier approaches to modeling durations of states in a Markov chain such as the approach by Vaseghi [14] in which state transition probabilities are conditioned 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. 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 (for example, see the complex function formed by combining multiple simple Gaussian functions in Figure 10).

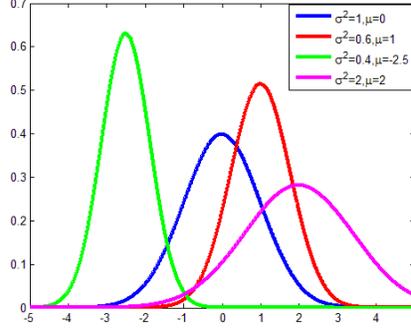


Figure 10 A complex function formed by combining multiple Gaussian functions.

3.6 Startup Trigger Information

Most pervasive computing mining algorithms generate a static model, by assuming that once the desired patterns from the sensor data have been learned, no changes need to be applied to maintain the model over time. The challenge is heightened when the underlying data source is dynamic and the patterns change. In the case of smart environments, as we know, humans are likely to change their habits and activity patterns over time depending on many factors, such as social relations, seasonal and weather conditions and even emotional states. Therefore, a static model cannot serve the purpose of a long-term solution for a smart environment. Instead, we need to find a way to adapt to the changes that occur over time.

As mentioned earlier, there are two options for detecting changes in discovered patterns. The fast explicit request option allows users to highlight a number of patterns to be monitored for changes; and the smart detection option automatically looks for potential changes of all patterns, based on a regular mining schedule.

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:

$$Q = (Q + \alpha * r) \quad (3)$$

In the above equation, r denotes the evidence value (predefined, e.g. 0.5), and $\alpha \in [0,1]$ denotes the learning rate which is a small value to imply a gradual history-preserving nature of learning as there is no strong evidence or user guidance that a change is a permanent change and not a temporary one. In order to avoid infinite

growth of potential, we allow value functions to grow only in the range of [-1...1]. Note that when updating the potential value, we do not differentiate between different events that comprise an activity; instead, we assign just a single value to an activity.

In the above equation, r denotes the evidence value (predefined, e.g. 0.5), and $\alpha \in [0,1]$ denotes the learning rate which is a small value to imply a gradual history-preserving nature of learning as there is no strong evidence or user guidance that a change is a permanent change and not a temporary one. In order to avoid infinite growth of potential, we allow value functions to grow only in the range of [-1...1]. Note that when updating the potential value, we do not differentiate between different events that comprise an activity; instead, we assign just a single value to an activity.

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

$$Q = Q - \frac{\varepsilon * \Delta t}{\theta} \quad (4)$$

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 to adapt to changing environment in smart detection method. The effect of the decay function is compensated through compensation effect in a way that the potential value remains bounded.

Explicit Request

In this 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 at 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 one.

In the case where structure is preserved (Figure 11), 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, the PAM algorithm reports the list of changes that can be accepted or rejected by the user.

```

Data: Activity  $a$ , Activity file, set of parameters
Result: detected changes in  $a$ 

// structure preserved
 $X_1 = \{$  all changed versions of  $a$  that preserve structure  $\}$ 

 $A_x = \{$   $period_x, duration_x, starttime_x, triggers_x$   $\}$ 
 $\theta_y =$  difference confidence for  $attribute_y \in A_x$ 

// detect changes that preserve structure
slide a window  $\omega$  of length  $|a|$  over all data
 $X' = \{$  all found sequences with the same structure as  $a$  $\}$ 

foreach  $a' \in X'$  do
| compute all  $y \in A_x$ 
end

foreach  $a' \in X'$  do
| foreach attribute  $y'_a$  in  $A_a$  do
| | if  $|y'_a - y_a| > \theta_y$  then
| | |  $X_1 = X_1 \cup a'$ 
| | end
| end
end

return  $X_1$ 

```

Figure 11 Detecting changes when structure is preserved.

Smart Detection

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 (Equation 3).

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.

In order to find out when a changed pattern replaces the original pattern, we can use the following analysis. If we denote the original pattern as O and the modified version of the pattern as M , then we can calculate the number of times the decay function should be applied, in order for O to be dominated by M . Assume at time t_i , pattern M is discovered for the first time and its potential value is assigned an initial value of Q_i^M . Also consider that the potential value for pattern O is initially Q_i^O . After time t_i , the decay function will periodically decrease the value of both patterns, while Q_i^M also increases each time M is observed. The potential value for pattern O , Q_u^O , after j applications of the decay function and at time t_u , will be:

$$Q_u^O = Q_i^O - \frac{\varepsilon * \Delta t}{\theta} \quad (5)$$

We also know that in order for the original pattern to be perfectly forgotten, its value function should be below an activity threshold, i.e. $Q_u^O < \sigma$. Substituting Equation 5 into $Q_u^O < \sigma$ leads to:

$$\frac{Q_i^O - \sigma}{\varepsilon} < j \quad (6)$$

The above inequality shows the minimum number of times that the decay function should be applied to a pattern before it's forgotten. At the same time, if we consider l observation due to regular mining sessions, Q_i^M will be changed as following:

$$Q_u^M = Q_i^M + l\alpha_o r_o - j\varepsilon \quad (7)$$

In order for Q_u^M to have a value greater than the activity threshold, we require that $Q_u^M > \sigma$. If we consider ΔT as an arbitrary time interval, and p as the period of regular mining (e.g. every week), then l can be defined in terms of m and ΔT , as $\Delta T/p$. Substituting Equation 7 into $Q_u^M > \sigma$ and considering ΔT and p leads to:

$$p < \frac{\alpha_o r_o * \Delta T}{\sigma + j\varepsilon - Q_i^M} \quad (8)$$

Equation 8 shows how frequently the old patterns will be replaced by new patterns.

4 Experimentation

Our goal is to develop a sensor mining model that unifies frequent and periodic pattern mining for pervasive computing applications. We desire that the model, as is essential for these applications, adapt to changes in those patterns over time and

automatically discover startup triggers and necessary temporal information. Here we evaluate our model using synthetic and real data collected in CASAS, our smart environment testbed located at Washington State University.

4.1 FPPM Evaluation

To provide a controlled validation of the FPPM algorithm, we implemented a synthetic data generator that simulates sensor 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 FPPM on real world data, we tested it on data obtained through sensors located in one room of the CASAS smart workplace environment. This physical testbed is equipped with motion sensors, rug sensors, light sensors, and controllers for the lamps (Figure 12).

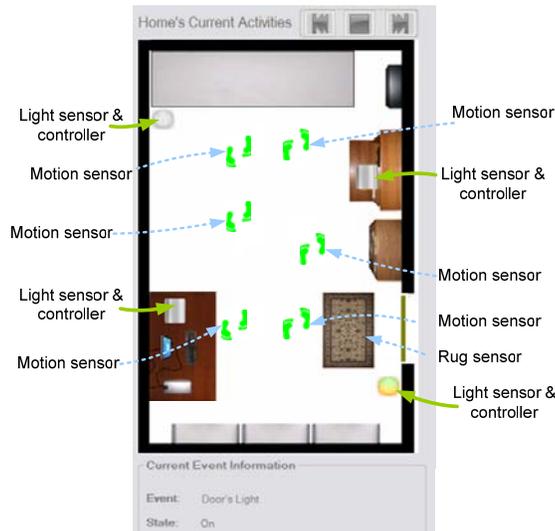


Figure 12 Testbed floor plan and sensor layout.

We initially generated one month of synthetic data that contained events for six activities. Five of the activities were event sequences of length two and the remaining activity was an event sequence of length three. One activity included an “open the door” trigger at the end and another included an “open the door” trigger in the middle of the sequence. The periods for these activities ranged from two to five hours. The devices and sensors that were simulated in this scenario were the same as those installed in WSU’s AI Lab. Details of the six simulated activities are listed here by start time, period, and components events with a duration of five minutes imposed for each event.

Our expectation was that FPPM would correctly identify all six activities with their corresponding periods, triggers, start times, and durations. In fact, FPPM was able to

find all of the activities with their correct periods. In addition, FPPM identified “open the door” trigger where it existed and accurately divided the sequence with a trigger in the middle into two subsequences where the trigger initiated the second subsequence. 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. For example, activity 1 had a compression rate of 6.7 instead of 8.0 and activity 5 had a compression rate of 6.0 instead of 6.6. However, in some cases the compression rates were higher than expected due to generation of similar random events. For example, split activities of activity 4 had a compression rate of 4.8 instead of 4.0. In other cases, the compression rates were as expected with minimal differences.

Table 2 Sample generated activities.

	Start Time	Period (hourly)	Events
1	13:00	2	DoorLight ON, LeftDeskLamp ON
2	13:30	3	WhiteboardLight ON, OpenDoor
3	14:25	5	RightDeskLamp ON, WhiteboardLight ON
4	14:55	2	LeftDeskLamp ON, OpenDoor, RightDeskLamp ON
5	15:35	3	LeftDeskLamp ON, WhiteboardLight ON
6	15:55	3	RightDeskLamp ON, DoorLight ON

In addition to these synthetic data experiments, we also tested FPPM on real smart environment data. Because we needed to validate that the discovered patterns align with what we knew existed in the data, we recruited a participant to execute a simple script in the smart workplace environment. The participant moved through the environment shown in Figure 5 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 FPPM then split into two subsequences. The light sensor also occasionally fired in the lab after the desk light was turned on. Despite these issues that occur in a real-world situation, FPPM 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 FPPM 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 2). 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).

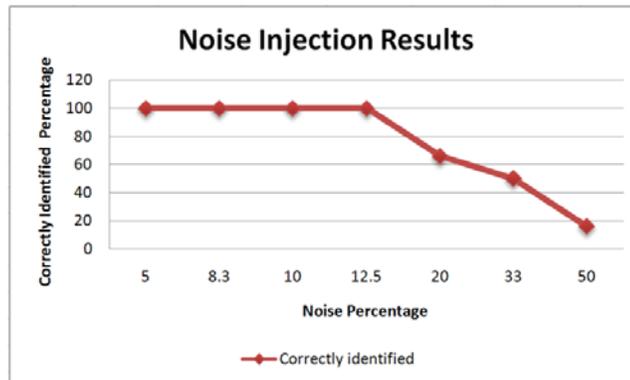


Figure 13 Noise injection results.

As it can be seen in Figure 13, FPPM 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.

4.2 PAM Evaluation

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 FPPM. After FPPM 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 7 minutes, instead of 5 minutes. PAM detected the changes accordingly by finding a new duration of 6.44 minutes, which is quite close to the actual 7 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 13:00 to 13:30, and structure changes (omission or addition).

In the next step, we tested PAM on real world data using our smart environment testbed. A volunteer participant entered the room and executed two different activities:

- Turn on right lamp(1 min), perform random actions
- 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, AAM detected some changes in start time. This is another correct finding by AAM. 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.

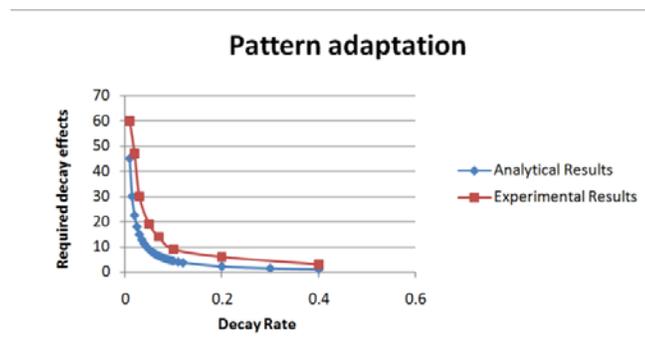


Figure 14 Changes in decay rate.

We also empirically validated our theoretical analysis to see how fast original patterns will be replaced by modified versions. To evaluate this, we designed an experiment in which we generated two sets of synthetic data similar to the first experiment. We then validated the adaptation capability for different decay values and

different initial value function (see Figure 14). Our findings are consistent with our expectation, validating that PAM can successfully adapt to resident changes even in real-world data.

5 Conclusions

In this paper, we introduced a new adaptive mining framework to detect sequential patterns in sensor event data and to adapt to changes in the underlying model. This framework uses the FPPM algorithm to discover frequent and periodic event patterns; and FPPM is complemented by using the PAM algorithm, which detects changes in the discovered patterns. Our framework also detects startup triggers in addition to temporal information. We presented the results of our experiments on both synthetic and real data. In our ongoing work, we plan to extend the FPPM method to include discovery of parallel activities, besides detection of abrupt changes. We also intend to discover additional types of contextual information that allow the model to better generalize over discovered sequences.

References

1. G.D. Abowd and E.D. Mynatt. Designing for the human experience in smart environments. In *Smart Environments: Technology, Protocols and Applications*, pages 153-174, 2005.
2. R. Agrawal and R. Srikant. Mining Sequential Patterns, Proc. 11th Int'l Conf. Data Engineering, pp. 3-14, 1995.
3. C. Bettini, S.X. Wang, S. Jagodia, and J.-L. Lin. Discovering Frequent Event Patterns with Multiple Granularities in Time Sequences, *IEEE Transactions on Knowledge and Data Engineering*, vol. 10, no. 2, pp. 222-237, 1998.
4. T. Fawcett and F. Provost. Activity Monitoring: Noticing Interesting Changes in Behavior, Proc. Fifth Int'l Conf. Knowledge Discovery and Data Mining, S., pp. 53-62, 1999.
5. Gaber, M. M., Krishnaswamy, S., and Zaslavsky, A., Adaptive Mining Techniques for Data Streams Using Algorithm Output Granularity, *The Australasian Data Mining Workshop (AusDM 2003)*, Canberra, Australia.
6. S. Helal, W. Mann. The Gator Tech Smart House: A programmable pervasive space. *IEEE Computer*, 38(3):50-60, 2005.
7. S. Laxman and P.S. Sastry. A survey of temporal data mining, *Sadhana* Vol. 31, Part 2, pp. 173-198, 2006.
8. C.H. Lee, M.-S. Chen, and C.-R. Lin. Progressive pattern miner: An efficient algorithm for mining general temporal association rules. *IEEE Transactions on Knowledge and Data Engineering* 15: 1004-1017, 2003.
9. W Lin, SA Alvarez, C Ruiz - Efficient Adaptive-Support Association Rule Mining for Recommender Systems. *Data Mining and Knowledge Discovery*, 2002 - Springer.
10. H. Mannila and H. Toivonen. Discovering Generalised Episodes Using Minimal Occurrences, Proc. Second Int'l Conf. Knowledge Discovery and Data Mining, pp. 146-151, 1996.
11. Rissanen, J. Modeling by shortest data description. *Automatica*, 14:465--471, 1978.

12. John F. Roddick, Myra Spiliopoulou. A Survey of Temporal Knowledge Discovery Paradigms and Methods, IEEE Transactions on Knowledge and Data Engineering, vol. 14, No. 4, 2002.
13. Sutton, R.S. On the significance of Markov decision processes . Artificial Neural Networks -- ICANN'97, pp. 273-282. Springer.
14. S.V. Vaseghi. State duration modeling in hidden Markov models. Signal Processing, 41(1):31-41, 1995.
15. V Vainio, A.-M., Vanhala, J. Continuous-time Fuzzy Control and Learning Methods. ISIT 2007. October 2007. Sydney, Australia.
16. G. M. Youngblood and D. J. Cook. Data mining for hierarchical model creation. IEEE Transactions on Systems, Man, and Cybernetics, Part C, 2007.