Itemset-based Mining of Constraints for Enacting Smart Environments

Viktoriya Degeler, Alexander Lazovik
Distributed Systems Group, Johann Bernoulli Institute
University of Groningen, The Netherlands
Email: {v.degeler, a.lazovik}@rug.nl

Francesco Leotta and Massimo Mecella
Dipartimento di Ingegneria Informatica, Automatica e Gestionale
SAPIENZA Università di Roma, Italy
Email: {leotta, mecella}@diag.uniroma1.it

***Abstract*—In order to automatically control the environment, smart systems should have sufficient rules, which describe expected system's behavior. While such rules may be added manually, usually this requires considerable efforts, often surpassing those that users are willing to spend to setup the system. In this paper, we propose a novel technique to mine such rules automatically, given a sensor log from the environment. In particular, we mine itemsets, but we consider abnormal drops in the frequency of variable state combinations w.r.t. the frequency of their subsets, which represent undesirability of these combinations. We evaluate the technique both on simulated and real datasets, showing that the approach is effective and promising for further extensions.**

I. Introduction

Systems for smart environments aim at recognizing needs and wishes of users, and at automatically controlling the cyber-physical space in order to satisfy them. However, in order to perform correct actuation actions, i.e., to change the environment in a desired way, the smart system must have sufficient information: the expected reactions to certain situations, the environmental constraints that must be satisfied, the expected goals to be accomplished, together with the services (i.e., available actions) to accomplish such goals, etc. As an example, in smart systems based on planning techniques and service-composition approaches (e.g., [1], [2]), the required information includes the description of available actions/services, the initial state and the final goals to be achieved by performing these actions. Conversely, in reactive rule-based systems (e.g., [3], [4]), the required information includes available sensor states, constraints over the environment, and current events (e.g., changes of sensor readings) to which the system should react in order to ensure the satisfaction of all constraints.

Such behavior rules may be entered manually into the system, or be based on pre-generated templates and domain knowledge. The manual creation of rules may be a viable choice for newly constructed smart systems; the manual addition of such rules is implementable, through appropriate user interfaces [5], in smart homes, where such rules inherently have high degree of legitimacy and correctness, as they by-creation represent the end-user desires. However, the efforts to create such rules manually are in most cases much larger than what users are willing to spend, even if pre-defined rule templates are taken into account. In larger environments, e.g., offices, smart residences for elderly persons, etc., in which the amount of rules is huge, such efforts are overwhelming and it is not guaranteed that manual insertion of rules will not contain internal contradictions, leading to unsatisfiable situations. Therefore, a promising approach is to obtain the rules through (fully- or semi-)automated learning, i.e., by collecting previous observations and inferring the rules from them. In this work, we aim at inferring rules automatically from sensor logs. The inferred rules may then be checked by a domain expert, and augmented or corrected, if required.

In this paper, we propose a novel technique to mine rules by using a variation of the Apriori algorithm [6]. We mine rules in the form of environmental constraints, i.e., we aim at finding restricted combinations of environmental values. In order to find them, we aim at detecting a sudden abnormal drop in frequency in logs of a certain variable combination w.r.t. its subsets. Such an abnormal drop may represent the fact that this set of values is undesirable, and the respective constraint should be created. Such mined constraints can be then directly used in reasoning systems, e.g., similar to the one described in [3], which are based on constraint satisfaction principles.

The remaining of this paper is as it follows. In Section II, we introduce preliminary definitions and the conceptual framework for our work. Then Section III describes the proposed technique, and Section IV validates it both on synthetic data, and on real ones based on a case study / living lab. Section V provides a discussion about relevant related work. Finally, Section VI concludes the paper by outlining promising future developments.

II. Preliminaries

Let $V = \{v_1, \ldots, v_n\}$ be a set of variables, each of them varying over a finite domain $D^{v_i} = \{d_1^{v_i}, \ldots, d_{n_i}^{v_i}\}$, describing the current state of the smart environment. Variables in V can be classified as either uncontrollable or controllable; uncontrollable variables are bound to *sensors* whereas controllable ones are bound to *actuators*. We assume values for both categories of variables to be accessible (at least during the mining phase); if for some actuators the assigned values cannot be directly acquired, we assume the availability of a set of sensors providing enough information to infer their current states (in such case a corresponding *virtual* sensor may be added).

Following [3], the expected behavior of the smart space automation system is defined as a set of rules, which must be satisfied at any particular instant of time. The rules are defined as predicate logic formulas where atoms take the form $v_i = d_k^{v_i}$ or $v_i \neq d_k^{v_i}$, with $d_k^{v_i} \in D^{v_i}$; well-formed formulas are obtained by composing atoms with logical connectives and quantifiers, such as $\neg$, $\wedge$, $\vee$, $\Rightarrow$, and $\Leftrightarrow$. Rules represent dependencies between variables and can be easily represented in the form of either allowed or forbidden assignments. As an example, the reader should consider a rule $Jack.location = room1 \Rightarrow room1.lamp = on$; the assignments allowed by this rule for the variables $(Jack.location, room1.lamp)$ are the following: $(\neg room1, on), (room1, on), (\neg room1, off)$. The very same rule can be written in terms of its forbidden assignment $\neg(Jack.location = room1 \wedge room1.lamp \neq on)$. It is easy to notice that representing rules as restrictions (or *constraints*) requires much less explicit statements, than if the rules were represented as possibilities, as environments are usually much more *permissive* than *restrictive*. Therefore we opted to mine rules in the form of *constraints* over the environment.

By adopting the terminology of the data mining literature, the assignment $v_i = d_k^{v_i}$ of a single variable to a single value of its domain will be referred to as an *item*. An *itemset* C is a combination of items such that no variable is seen more than once in it, i.e., $C = \{v_i = d_k^{v_i} \mid \forall i \neq j : v_i \neq v_j\}$. Constraints, i.e., forbidden variable assignments, are returned by the technique presented in Section III-A in the form of itemsets. A *situation* S is an itemset assigning a value to all the variables in V; a situation represents the snapshot of the smart space in a given instant of time $t(S)$.

The input to our algorithm is a dataset $\mathcal{T} = \{< S_1, w_1 > \cdots < S_{|\mathcal{T}|}, w_{|\mathcal{T}|} >\}$[1]; it is obtained by generating, every time the state of a single variable in V changes, a new couple $< S, w >$, where S is a situation and w is a weight assigned to it. The weight w is a real number, which defines how much S is significant inside the dataset.

When new situations are not added to the dataset at a regular pace, being them added after the arrival of a new measurement, they should be accounted proportionally to the time $\Delta(S)$ they remain valid as current state of the system. If two situations S_j and S_{j+1} are added one after the other to $\mathcal{T}$, then $\Delta(S_j) = t(S_{j+1}) - t(S_j)$. Following this intuition, the weight is calculated as follows:

$$w = \frac{\Delta(S)}{\alpha} \text{ with } \alpha = \frac{t(S_{|\mathcal{T}|}) - t(S_1)}{|\mathcal{T}| - 1} \tag{1}$$

Conversely, if new situations are added at a fixed pace, w can be set to an arbitrary real value β (e.g., during our evaluation over the real dataset we used $\beta = 1$); in this case two consecutive situations may be equal in terms of variable values as they represent the very same environment state.

Given an itemset C and a dataset $\mathcal{T}$, we define $\mathcal{T}^C = \{< S_i, w_i > \in \mathcal{T} \mid C \subseteq S_i, i \in [1, |\mathcal{T}|]\}$ as the set of couples in $\mathcal{T}$ whose situations contain C. The weight of C inside $\mathcal{T}$ is defined as follows:

$$w(\mathcal{T}^C) = \sum_{< S_i, w_i > \in \mathcal{T}^C} w_i \tag{2}$$

If $w_i = \beta$ for every couple in $\mathcal{T}$, the definition reduces to $w(\mathcal{T}^C) = \beta|\mathcal{T}^C|$.

[1]Given a set A, we denote with $|A|$ the cardinality of A.

III. Rule Learning

The mining technique proposed here is a variant of the seminal Apriori algorithm [6] employed for mining *frequent* itemsets.

Definition 1 (Frequent Itemset)**.** *Given an itemset C, its* support *in $\mathcal{T}$ is defined as the fraction $Supp(C) = \frac{w(\mathcal{T}^C)}{W}$, where $W = \sum_{i=1}^{|\mathcal{T}|} w_i$ is the sum over the entire dataset $\mathcal{T}$ of the weights w_i. An itemset C is told to be frequent if its support is above a minimum threshold value.*

Many variations to the original Apriori algorithm have been proposed in the literature in order to meet the requirements of particular applications. In some cases, the discovery of *rare itemsets*, instead of frequent ones, may be particularly useful. If, in the ideal case, an itemset correspondent to a forbidden assignment should never happen, in practice, this is not the case, and forbidden combinations still occur due to erroneous sensor readings, non-immediate reaction of users and deviations from the standard behavior. An itemset is usually told to be rare if it has a very low support. Unfortunately, fixing a very low support threshold for the Apriori algorithm is not enough to find rare itemsets, as it would generate a huge number of redundant and unnecessary itemsets; this is known as the *rare itemset problem*. Additionally the definition of rare itemset is application specific. In [7], a rare itemset is defined as "one which its frequency in the database does not satisfy the minimum support but appears associated with the specific data in high proportion of its frequency". In [8], a rare itemset is somewhat *interesting* only if it represents an exception (e.g., there are many vegetarians and many persons with cardiovascular diseases, but very few people are both); thus, the final goal of the proposed Apriori-Rare algorithm is to compute *minimal rare itemsets* (mRI), which are those itemsets that are rare but whose proper subsets are frequent.

Other works, e.g., [9], [10], point out that it may be more interesting to find combinations of items that are not necessarily frequent, i.e., they may have low *support* value, but a high *confidence* rate, i.e., high probability of being together in the same transaction. This consideration is applicable in our scenario. As an example, the reader should consider a situation when a person goes to the bathroom, opens a bathroom cabinet, takes a toothbrush, turns on the water tap, etc. The amount of minutes people spend doing this activity is very low, comparing to the full log of the smart space, thus this situation has low support and cannot be regarded as "frequent". However, it can be said with very high confidence that if a person is in the bathroom, and the toothbrush is taken, then the water tap is always turned on. A constraint to describe the above case could be $\neg(Jack.location = bathroom \wedge Jack.toothbrush = taken \wedge bathroom.watertap = off)$.

Our definition of "interesting" is somewhat orthogonal to the above mentioned ones. Instead of fixing a threshold in order to distinguish frequent from rare itemsets, a dynamic threshold is computed for each itemset aimed at detecting a significant drop in its weight with respect to its subsets; we will refer to such itemsets as *restricted*.

It can be trivially shown that the support of an itemset is always smaller or equal than the support of each of its proper subsets. We formalize this drop by using the concept of *decline*:

Definition 2 (Decline)**.** *Given two itemsets C and C_p, where C is obtained by adding a single item to C_p, i.e., by assigning a value to a variable v_i not appearing in C_p (C_p is told to be* parent *of C), we define the* decline *of C with respect to C_p as* $\frac{w(\mathcal{T}^C)}{w(\mathcal{T}^{C_p})} \leq 1$.

Decline is a natural phenomenon as we are constraining the variable v_i to assume a specific value $d_j^{v_i} \in D^{v_i}$. In order to detect the amount of decline that is abnormal, we can define the *normalized decline* as follows:

$$\frac{w(\mathcal{T}^C)}{p(d_j^{v_i})w(\mathcal{T}^{C_p})} \leq 1 \tag{3}$$

Here $p(d_j^{v_i})$ represents the probability of a variable v_i to be in a state $d_j^{v_i}$. We need this probability, as many variables in smart environments have non-uniform distribution. For example, during a working day, a PC has much bigger chance to be turned on, than being off. Specifically, for variables with uniform distribution $p(d_j^{v_i}) = \frac{1}{|D^{v_i}|}$. For a variable showing a non-uniform distribution, instead, if $\mathcal{T}^{d_j^{v_i}} = \{< S_i, w_i > \in \mathcal{T} \mid (v_i = d_j^{v_i}) \in S_i, i \in [1, |\mathcal{T}|]\}$, we have:

$$p(d_j^{v_i}) = \frac{w(\mathcal{T}^{d_j^{v_i}})}{W} \text{ where } w(\mathcal{T}^{d_j^{v_i}}) = \sum_{<S_i,w_i>\in\mathcal{T}^{d_j^{v_i}}} w_i \tag{4}$$

If the normalized decline of an itemset C is close to 1, this means that the combination of variable assignment it represents is perfectly admissible for the final users; an *abnormal* decline (with much smaller closeness values) may be an indicator that C is an undesired, restricted, combination. As, in general, the very same itemset C can be obtained by adding a variable assignment to a set $\mathcal{C}_p$ of different parent itemsets, we define a restricted itemset as follows:

Definition 3 (Restricted Itemset)**.** *An itemset C is told to be restricted if its normalized decline with respect to each of the parent itemsets in $\mathcal{C}_p$ is less than a predefined threshold $r \in$* $[0, 1]$ *called "restriction rate".*

As example, assume we have a small environment with three boolean variables: PC with weights $w(PC = 0) = 80$, $w(PC = 1) = 137$; LCD with weights $w(LCD = 0) = 145$, $w(LCD = 1) = 72$, and chair PR[essure] with weights $w(PR = 0) = 92$, $w(PR = 1) = 125$. The weights of combined itemsets $w(\{PC = 1; LCD = 0\}) = 65$ and $w(\{PC = 1; LCD = 1\}) = 72$ both represent normal decline rate, and therefore are considered permitted. On the other hand, the weight of itemset $w(\{PC = 0; LCD = 1\}) = 0$ clearly indicates that such situation is restricted and will generate a constraint $\neg(PC = 0 \wedge LCD = 1)$, which represents physical constraint that LCD cannot be turned on if the PC is off. Another situation $w(\{LCD = 1; PR = 0\}) = 2$ is also restricted, because the decline rate from both parents (with weights 72 and 92) is still large enough, even though the situation happened for a couple of minutes. This constraint shows that there is a big preference to turn off a monitor if no-one is sitting in front of the PC.

Algorithm 1 Find restricted variable sets

```
1:  function findConstraints ($\mathcal{T}$ - dataset; $r$ - restriction rate)
2:  for all $d_j^{v_i} \in D^{v_1} \cup \cdots \cup D^{v_n}$ do
3:      $p(d_j^{v_i}) \leftarrow \frac{|v_i = d_j^{v_i}|}{|\mathcal{T}|}$
4:  end for
5:  $\mathcal{R} \leftarrow \{$1-itemsets $S$ s.t. $\frac{w(\mathcal{T}^S)}{|D^{v_i}|*|\mathcal{T}|} < r\}$
6:  $\mathcal{I} \leftarrow \{$all 1-itemsets$\}\backslash\mathcal{R}$
7:  while $|\mathcal{I}| > 0$ do
8:      $\mathcal{C} \leftarrow getSetOfChildren(\mathcal{I})$
9:      $\mathcal{I} \leftarrow \emptyset$
10:     for all $C \in \mathcal{C}$ do
11:         $W_P \leftarrow \emptyset$
12:         for all $P \in parents(C)$ do
13:             $d_j^{v_i} \leftarrow addedValue(P, C)$
14:             $W_P \leftarrow W_P \cup \{w(\mathcal{T}^P)p(d_j^{v_i})\}$
15:         end for
16:         if $\frac{w(\mathcal{T}^C)}{min(W_p)} < r$
17:             $\mathcal{R} \leftarrow \mathcal{R} \cup \{P\}$
18:         else if $w(\mathcal{T}^C) > 0$
19:             $\mathcal{I} \leftarrow \mathcal{I} \cup \{P\}$
20:         end if
21:     end for
22: end while
23: return $\mathcal{R}$
```

A. Algorithm description

Our approach to mine constraints in a smart environment is listed in Algorithm 1. The algorithm takes as input a dataset $\mathcal{T}$ and the desired *restriction rate* r, and it returns as output a set $\mathcal{R}$ of restricted itemsets representing constraints. As a first step, the distribution p of domain values is calculated (lines 2-4).

Following the original Apriori algorithm, at every iteration, the algorithm processes itemsets with increasing size stored in the set $\mathcal{C}$. At the k-th iteration, $\mathcal{C}$ contains only k-itemsets (itemsets containing exactly k items) obtained by combining (using the $getSetOfChildren$ function at line 8) those $(k-1)$-itemsets, contained in the set $\mathcal{I}$, that was not declared as restricted at the $(k-1)$-th iteration. Given a k-itemset C, its parents are those $(k-1)$-itemsets contained in $\mathcal{I}$ such that all their items are contained in C. Thus, every itemset has a number of parents equal to its size. As an example, suppose $\mathcal{I} = \{\{v_1 = d^{v_1}, v_2 = d^{v_2}\}, \{v_1 = d^{v_1}, v_3 = d^{v_3}\}, \{v_2 = d^{v_2}, v_3 = d^{v_3}\}\}$, then $getSetOfChildren(\mathcal{I}) = \{\{v_1 = d^{v_1}, v_2 = d^{v_2}, v_3 = d^{v_3}\}\}$. Before iterations begin, $\mathcal{R}$ is initialized with all restricted 1-itemsets, and $\mathcal{I}$ is initialized with all the possible 1-itemsets, except those that are restricted. We regard 1-itemset as restricted if its closeness to the uniformly distributed values is less that the restriction rate. During each iteration the set $\mathcal{I}$ is immediately emptied and will be filled with the itemsets to be expanded during the successive iteration; the algorithm stops as soon as, after the conclusion of an iteration, $\mathcal{I}$ is empty (see line 7).

During a specific iteration, the weight of each itemset $C \in \mathcal{C}$, is compared to a dynamically computed threshold dependent on the value of r. Restriction rate represents the maximum decline value for considering an itemset as restricted. As a C has, in general, more than a single parent, the abnormal decline should be registered for all its parents in order to declare it as restricted. To this aim, it is sufficient to compare r with the maximum value of decline calculated (see line 16).

At every iteration, all the itemsets in $\mathcal{C}$ that satisfy the condition at line 16 are put into the set $\mathcal{R}$ of constraints (see line 17). Itemsets in $\mathcal{C}$ that do not satisfy the same condition are put into the set $\mathcal{I}$ (see line 19), unless their weight is equal to 0, which means that the corresponding combination of values is never seen in the dataset, so there is no point in trying to expand it further. This happens when the decline difference of all itemset's parents never exceeded the restriction rate, which means decline was always within expected limits until finally reaching 0.

IV. Validation

We assessed the performance of the proposed algorithm by running different experiments on both synthetic and real datasets. Tests on the synthetic dataset highlight how much the accuracy changes by tuning different parameters (see Section IV-A). The evaluation on the real dataset shows the effectiveness of the algorithm in a realistic application scenario.

During the evaluation, accuracy results represent the similarity of original rules and mined ones by constructing the full truth table for all n variables, and counting the combinations of sensor values in the table returning the same result ("allowed" or "restricted") for both original and mined rules.

A. Validation on Synthetic Datasets

As it is very difficult to obtain datasets large enough to evaluate algorithms in a wide range of input configurations, a first quantitative evaluation has been conducted on a configurable synthetic dataset.

For these experiments we used a generation tool producing a dataset $\mathcal{T}$ consisting of $|\mathcal{T}|$ situations. A situation binds n boolean variables; however, it is always possible to transform a discrete variable into a set of boolean variables. As previously discussed, each of the produced situations is supposed to have an associated weight; as the generation tool generates situations at a constant pace, the weight is always equal to $\beta = 1$. A parameter p_c represents the probability that at each discrete time step the state of the environment changes; p_c reflects how much the simulated environment is dynamic.

The synthetic dataset is built by trying to respect a randomly generated set of n_c constraints; each constraint binds up to 4 variables, as this is a reasonable limitation for a real environment. Violations to rules are accepted with a probability p_v, which represents how much the inhabitants of the simulated environment are prone to deviate from their usual behavior (the one we aim to mine). Finally a parameter p_p represents the probability that a violation is maintained after a single step; after k steps this probability becomes p_p^k.

At every step with probability p_c a change is applied to a variable; supposing this change introduces n_v new violations and solves n_s previously introduced violations, the probability of adding this change to the dataset is equal to $p_v^{n_v} \cdot \prod_{i=1}^{n_s}(1 - p_p^{k_i})$, where k_i denotes the number of steps since i-th violation was introduced.

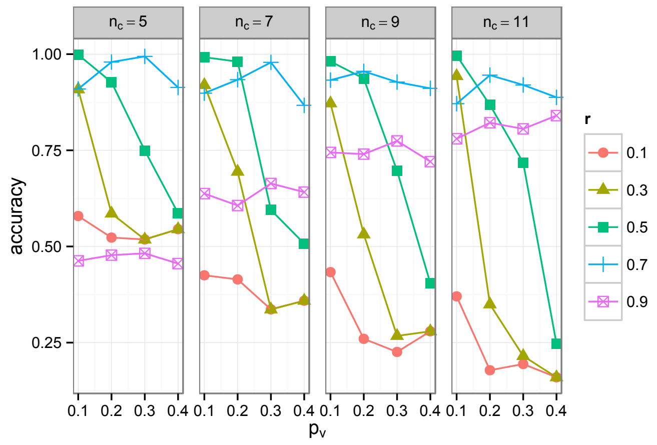
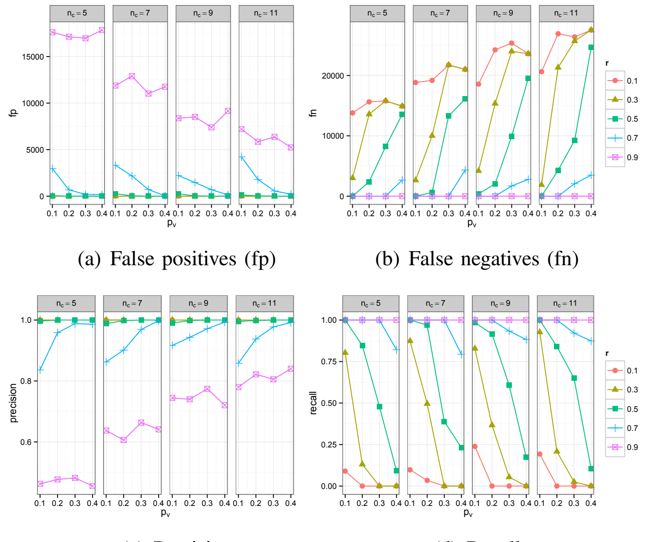


Fig. 1. Accuracy of the algorithm in terms of correctly interpreted situations.



(a) False positives (fp) (b) False negatives (fn)

(c) Precision (d) Recall

Fig. 2. Analysis of the errors introduced by the algorithm.

Figure 1 and Figure 2 represents the results obtained by applying our algorithm to different significant parameters of both the algorithm and the generation tool. Each facet represents the performance of the algorithm with a different number n_c of constraints. The performance of the algorithm, with respect to the probability p_v of adding a violation, is calculated for five different values of the r parameter (see Section III-A), namely: 0.1, 0.3, 0.5, 0.7 and 0.9. Each single condition has been repeated 20 times using different values of the p_p parameter[2]. All the conditions generate a number of situations $|\mathcal{T}| = 10000$ by keeping the probability p_c of changing the environment fixed to 0.8 and the number n of variables fixed to 15.

Parameter r tells the algorithm how much the actual decline should differ from the expected one in order to declare a constraint as restricted. Figure 1 shows how for low values of p_v, putting $r = 0.5$ provides the best results, with an accuracy

[2]During our evaluation we noticed that p_p has a small influence on the achieved accuracy.

greater than 75% when p_v is less than 0.2. On the other hand, the higher is the value of p_v, the higher will be the number of violations inside the dataset, thus, the higher will be the weight of the constraint inside the dataset; as a consequence, in order to declare an itemset as restricted we need r to have an high value.

Figure 2 thoroughly inspects how the errors committed by the algorithm are distributed. It can be noticed how, except for high values of the parameter r, the algorithm provides very good performances in terms of false positives, intended as situations allowed in the original dataset that are forbidden by the mined constraints; this means that the proposed algorithm does not introduce unwanted restrictions. From the point of view of false negatives, intended as situations forbidden in the original dataset that are allowed by the mined constraints, the performances of the algorithm quickly get worse for small values of r as either p_v or n_c increase. As a consequence of the previous considerations, small values of r increases the precision of the algorithm; conversely, high values of r increases the recall provided by the algorithm.

For small values of r, the mined rules are very close to the original set and this makes them useful if we aim at mining original constraints for enacting the smart environment. On the other hand, by increasing the value of r, the algorithm will over-fit the original data by producing a set of rules with an high cardinality that is very different from the set of constraints employed to generate the dataset. Nevertheless rules extracted with high values of r can be still employed for detecting unusual conditions of the environment without automatically perform any modification to the environment.

Even though results get worse as either p_v or n_c increase, putting $r = 0.5$ provides, overall the best results. Moreover, it is likely to expect that rules that the users want to be respected are not violated more than 15% of the time.

B. Living Lab Case Study

To evaluate the algorithm on real dataset, we used the dataset obtained from previous experiments for a small living lab. Here we briefly explain the design of the experiment and the results. The experiment was performed on the premises of the University of Groningen, where two identical working rooms were populated with sensors and sensor data was collected for three days. The original setup is described in details in [11]. Each room contains a desk with PC, LCD, two motion (PIRK, PIRM), two acoustic (AC, ACK), and two chair pressure (PR1, PR2) sensors. The goal of the original experiment was to use the rules of the environment to find and resolve sensor errors to improve activity recognition (AR) rate. Therefore the rules of environment were written manually. In this experiment we used the original sensor data (before sensor errors correction, as it was using manually written rules) and applied our algorithm to it to mine the rules automatically. We then used those mined rules and compared them with the original manually created ones. Note that original sensor data was quite noisy, with activity recognition rate from the original data being 80.9% in the first room and 76.3% in the second one. Sensor readings were collected once every minute during working hours. Three days of data from two different rooms with identical setup were combined to obtain a log of 2402 situations in total. Among those situations 199 (8.3%) minutes represented the full absense of people in a room; 344 (14.3%) situation represented presense of people in a room, but not in front of a PC or sitting at working desks; during 70 (2.9%) minutes in total meetings were held; 395 (16.4%) situations represented different types of paperwork, i.e. people working at a desk, but not with PCs; and 1394 (58.0%) minutes were dedicated to working with PCs. Note that several sensor value combinations can sometimes represent the same type of a situation, for example working with PC may happen with background motion sensor showing motion or not.

TABLE I. MANUAL VS MINED CONSTRAINTS (SHOWN AS ITEMSETS)

Original	Mined
LCD=T PC=F	PC=F
LCD=T PR1=F PR2=F	AC=T PR2=T
LCD=T PIRK=F	ACK=F PR2=T
PIRK=T PR1=F PR2=F	LCD=T PIRK=F
AC=T ACK=F	PIRK=F PR1=F PR2=T
AC=T PIRM=F	ACK=T PIRK=T PIRM=F PR1=F PR2=F
AC=F PR1=T PR2=T	AC=T ACK=T PIRK=F PR1=T

The original and mined constraints are shown in Table I. There are several interesting points to mention. For example, the algorithm found the constraint $\neg(PC = F)$. This constraint was not part of the manual rules. However, all data collection was done during working hours, and during all that time both PCs in both rooms were always turned on. That means that the found constraint represents the preference rule "During the working day PC should be turned on", which follows directly from actual users behavior, and therefore in some sense is more representative of the actual situation than manually written rules were. Some constraints, as $\neg(LCD = T \wedge PIRK = F)$ are contained in both the original and mined sets, while others, as $\neg(PIRK = T \wedge PR1 = F \wedge PR2 = F)$, are found as a part of more restrictive constraints, namely $\neg(ACK = T \wedge PIRK = T \wedge PIRM = F \wedge PR1 = F \wedge PR2 = F)$. The accuracy results were that for 256 combinations of the truth table, 185 were the same (72.3%), while 71 were different. Noteworthy, the obtained accuracy is comparable to that reported in [12], where Event-Condition-Action rules in smart environments are mined.

The less regular human behavior is the bigger the input dataset needs to be in order to not integrate spurious behavior into the results. As in our experimental setup users were instructed to execute routines in a predefined way, three days of recordings are enough in order to obtain good results. In order to perform experiments over a not specifically designed real dataset, the size of the dataset needs to be selected in order to represent the average behavior of the users.

V. RELATED WORK

Data mining techniques have been already employed in smart environments in order to infer human habits or unusual (e.g., dangerous) situations. Authors in [13] applies (point-wise) inter-transaction association rules (IAR) mining in order to detect *emerging* (i.e., unusual) behaviors; the first step consists of mining IAR rules from sensor logs, whereas emerging ones are identified, as a second step, during runtime by comparison. The real world data consisted of event logs from an array of state-change sensors. The SPUBS system [14] merges concepts from both data mining and workflow mining.

The proposed technique is mainly supervised and supposes the availability of a huge amount of labeled data. The CASAS project employs a pattern mining technique [15], [16] in order to discover human activity patterns. This method allows to discover *discontinuous patterns* by iteratively trying to compress the dataset. A pattern is defined as a set of events where order is not specified and events can be repeated multiple times. The data mining method employed for activity discovery is completely unsupervised without the need of manually segmenting the dataset or choosing window sizes.

All the aforementioned works differ from our technique as they all are *event-based* and mainly focused on the run-time recognition of activities. Conversely, we concentrate on discovering unusual states of the environment in order to generate *reactive rules* that are triggered in the case mined constraints are violated. From this point of view, the final goal is similar to that of the APUBS [12] system. APUBS mines simple Event-Condition-Action (ECA) rules, where an ECA rule takes the form "ON *event* IF *condition* THEN *action*". Still, the approaches are different, as while APUBS inherits from SPUBS a workflow-based approach, we do not impose any order between state changes. This allows us to mine human habits from a different point of view. Suppose, as an example, that users never use the washing machine and the dishwasher at the same time; this kind of rule cannot be mined using the approach employed by APUBS as it is not possible from any log to directly define a sequence relationship between the power switch events of the two devices.

VI. Concluding remarks

In this paper we have presented a novel technique for mining constraints in smart spaces. The key idea of our technique, based on the Apriori algorithm, is to consider abnormal drop of support of a certain variable combination w.r.t. its parents, as this may represent undesirability of this combination, and therefore it may be regarded by the system as an environmental constraint.

As a main contribution, differently from previous works, which extract ordered workflow of events representing human habits, the proposed approach focuses on the state of the environment in order to mine undesired situations that can be used to extract reactive rules for smart environments.

Our evaluation demonstrates that the algorithm indeed discovers the desired constraints even if errors and undesirable situations are present in a sensor log. In future work, our aim is to continue evaluation with bigger and real datasets. In this sense, integrating ontologies of devices into the mining process would allow to extract restricted rules also taking into account the semantic relationships among devices.

This algorithm represents a first step towards the design of techniques aimed at mining complete user habits over the environment.

Acknowledgments. This work has been partly supported by the EU project GreenerBuildings (contract FP7-258888). The work of Viktoriya Degeler has been partly performed when a visiting internship student at Sapienza Università di Roma. The work of Francesco Leotta and Massimo Mecella has been also partly supported by AriSLA through the Italian project Brindisys, by Regione Lazio and FILAS through the "Progetto Sensoristica Interconnessa per la Sicurezza", and by SAPIENZA Università di Roma and DIAG through the grant "Assegni di Ricerca 2012".

References

[1] E. Kaldeli, E. U. Warriach, A. Lazovik, and M. Aiello, "Coordinating the web of services for a smart home," *ACM Trans. on the Web*, vol. 7, no. 2, 2013.

[2] M. Caruso, C. Di Ciccio, E. Iacomussi, E. Kaldeli, A. Lazovik, and M. Mecella, "Service ecologies for home/building automation," in *Proc. 10th International IFAC Symposium on Robot Control (SYROCO 2012).*

[3] V. Degeler and A. Lazovik, "Dynamic constraint reasoning in smart environments," in *Proc. 25th IEEE Int. Conf. on Tools with Artificial Intelligence (ICTAI 2013).*

[4] V. Degeler, L. I. Lopera Gonzalez, M. Leva, P. Shrubsole, S. Bonomi, O. Amft, and A. Lazovik, "Service-oriented architecture for smart environments," in *Proc. 6th IEEE Int. Conf. on Service Oriented Computing and Applications (SOCA 2013)*

[5] C. Di Ciccio, M. Mecella, M. Caruso, V. Forte, E. Iacomussi, K. Rasch, L. Querzoni, G. Santucci, and G. Tino, "The homes of tomorrow: Service composition and advanced user interfaces," *ICST Transactions on Ambient Systems*, vol. 11, no. 10-12, p. e2, 12 2011.

[6] R. Agrawal, R. Srikant *et al.*, "Fast algorithms for mining association rules," in *Proc. 20th Int. Conf. Very Large Data Bases (VLDB 1994).*

[7] H. Yun, D. Ha, B. Hwang, and K. Ho Ryu, "Mining association rules on significant rare data using relative support," *Journal of Systems and Software*, vol. 67, no. 3, pp. 181–191, 2003.

[8] L. Szathmary, A. Napoli, and P. Valtchev, "Towards rare itemset mining," in *Proc. 19th IEEE Int. Conf. on Tools with Artificial Intelligence (ICTAI 2007).*

[9] Y.-L. Cheung and A. W.-C. Fu, "Mining frequent itemsets without support threshold: with and without item constraints," *IEEE Trans. on Knowledge and Data Engineering*, vol. 16, no. 9, pp. 1052–1069, 2004.

[10] Y. S. Koh and N. Rountree, "Finding sporadic rules using apriori-inverse," in *Proc. 9th Pacific-Asia Conference on Advances in Knowledge Discovery and Data Mining (PAKDD 2005).*

[11] T. A. Nguyen, V. Degeler, R. Contarino, A. Lazovik, D. Bucur, and M. Aiello, "Towards context consistency in a rule-based activity recognition architecture," in *Proc. Int. Symposium on Ubiquitous Intelligence and Autonomic Systems*, 2013.

[12] A. Aztiria, J. C. Augusto, R. Basagoiti, A. Izaguirre, and D. J. Cook, "Discovering frequent user–environment interactions in intelligent environments," *Personal and Ubiquitous Computing*, vol. 16, no. 1, pp. 91–103, 2012.

[13] S. Lühr, G. West, and S. Venkatesh, "Recognition of emergent human behaviour in a smart home: A data mining approach," *Pervasive and Mobile Computing*, vol. 3, no. 2, pp. 95–116, 2007.

[14] A. Aztiria, A. Izaguirre, R. Basagoiti, J. C. Augusto, and D. J. Cook, "Automatic modeling of frequent user behaviours in intelligent environments," in *Proc. 6th Int. Conf. on Intelligent Environments (IE 2010).*

[15] P. Rashidi, D. J. Cook, L. B. Holder, and M. Schmitter-Edgecombe, "Discovering activities to recognize and track in a smart environment," *Knowledge and Data Engineering*, vol. 23, no. 4, pp. 527–539, 2011.

[16] D. Cook, N. Krishnan, and P. Rashidi, "Activity discovery and activity recognition: A new partnership," *Cybernetics, IEEE Transactions on*, vol. 43, no. 3, pp. 820–828, 2013.