Temporal Reasoning for Intuitive Specification of Context-Awareness

Unai Alegre Ibarra
University of Mondragon
Mondragon, Spain
Email: unai.alegre@alumni.mondragon.edu

Juan Carlos Augusto
Research Group on Development
of Intelligent Environments
Dept. of Computer Science, Middlesex University
London, UK
Email: j.augusto@mdx.ac.uk

Asier Aztiria
University of Mondragon
Mondragon, Spain
Email: aaztiria@mondragon.edu

***Abstract*—One of the most important challenges of the creation of intelligent environments is the specifications of what intelligent behaviours the system will exhibit. The processing of these situations can be computationally demanding. We report on the advances of the specification of a rule-based language which allows for the natural expression of situations of interest as those which occur on Intelligent Environments. The language focuses on quasi real-time situations and includes new temporal operators which allow a natural reference to time instants and to intervals. We explained how the system is implemented and how the system was validated within a Smart Office scenario.**

***Index Terms*—ambient intelligence, context awareness; temporal reasoning**

I. Introduction

There is always a tension within Computer Science for expressiveness and computational efficiency, and this naturally happens in the area of Intelligent Environments as well. With that in mind, we have been exploring different variations of simple logical languages which can be useful to naturally express ambient intelligence. A minimal reasoning system to handle causality introduced by Galton and Augusto [1], let us call it C, has been extended with uncertainty and applied to smart homes scenarios in the *MTPL* language [2]. On the other hand, temporal notions were in those languages pretty simple and some situations require richer temporal expressiveness to be of practical use. It is also desirable that intelligent environment systems have both reasoning and learning capabilities. The work presented in this paper provides an extension to the specification language which facilitates practical reasoning and prepares the system to be coupled with the learning system *LFUBS* [3]. This work focuses on the explanation of the extended reasoning system, *M*, which now allows more comfortable references to time and gives us more natural and comfortable tools to implement context-awareness. We will address the connection with *LFUBS* in another future paper. Instead we focus here on the following two key principles of the Intelligent Environments Manifesto [4]:

P1) to be intelligent to recognize a situation where it can help.
P7) to have autonomous behaviour.

The following sections explains the characteristics of *M* (Section II), some examples which help to understand the behaviour of the system being presented, (Section III), a current real scenario being used to validate the system, Section IV, and finally our conclusions (Section V).

II. Specification Language and Reasoning

We start with a simple system, *C*, presented in [1] which allowed to represent and reason with elegant simplicity the connections between causes and effects. The system had a well defined language and associated algorithm which we adopt as a departing point. We also noticed that richer temporal conditions were needed to deal with some scenarios more comfortably to make it of practical use. This revised system *M*, includes more precise and expressive references to conditions which have held for a while in the recent past.

The representation of systems will emphasize the dynamic nature of the scenarios we considered and the importance of time to represent context-awareness to develop ambient intelligence. The building blocks of this theory are an *atomic state set* S, a *rule set* R and an *event set* E, and the next sub-sections explain each of these in turn.

A. States

The set of all atomic states S is divided into two subsets, *independent* (S_I) and *dependent* (S_D), where $S_I \cup S_D = S$ and $S_I \cap S_D = \varnothing$. An independent state does not depend causally on other states holding at the same time. Each state can be either *true* or *false*:

$$[\neg]s$$

Where s is the state, and $\neg$ is the *boolean operator* for negation. In this context, brackets mean that the operator is optional. Absence of boolean operator evidences positive status (s). The status of a state and the time when this happens, is represented by the predicate:

$$HoldsAt([\neg]s, t)$$

Where t is the time measured in atomic time units and represented by a natural number ($t \in \mathbb{N}$). For a state s, when s is true it will be said that the state holds: $HoldsAt(s, t)$ and for state $\neg s$, it will be said that holds, when s is false: $HoldsAt(\neg s, t)$.

An important limitation of *C* is the difficulty to express conditions which must hold for a while in order for a given context of interest to be detected. Typical notations to represent durations range from temporal operators to explicit references to time intervals, see for example [5]. Here we choose the first route as it provides a more natural linguistic constructions. Usually temporal operators are based on the traditional 'G' operator representing that a certain condition is always true, and the dual 'F' operator representing that a certain condition is sometimes true. This notation has been replaced in the latest decades with $\square$ and $\diamond$ which we adopt here as well. One problem of these operators which limits their practical use in certain domains is their lack of precision, even if we consider their related 'always in the future' or 'always in the past', that does not tell us a great deal about where exactly those conditions held. To react to specific contextual circumstances in an intelligent environment we need to be able to refer to what had happened at specific in the immediate past or to specific times of the day. Hence we consider specializations of those operators which are more 'metric' in nature and allow us to say with more precision when exactly those relevant conditions held.

The sustained holding of a state throughout a certain duration of time in the past can be represented as follows:

$$[BP_{Op}][\neg]s$$

where BP_{Op} represents one of our 'bounded past' operators:

1) Immediate past, relative to present:

$$\boxminus_{[\mu]}$$

Square brackets used as a sub-index to a bounded past operator are used to represent an extension of time, same times relative to present, sometimes absolute. The sequence of states represented by $\boxminus_{[\mu]}[\neg]s$, referred as *Strong immediate past operator*, will be true if and only if $HoldsAt([\neg]s, \tau)$ for all possible τ where:

- $(t \dot{-} \mu) \leq \tau < t$, being t the present time expressed as absolute.[1]
- $\mu > 0$

An example is provided in section III-A.

2) Absolute reference to the past:

$$\boxminus_{[\alpha,\beta]}$$

Noticing that brackets surrounding α and β do not mean optionality, the state preceded by the bounded past operator $\boxminus_{[\alpha,\beta]}$, represented as $\boxminus_{[\alpha,\beta]}[\neg]s$, referred as *Strong absolute reference to the past operator*, is true if and only if $HoldsAt([\neg]s, \tau)$ for all possible τ where[2]:

- α and β represent absolute time.
- $\alpha \leq \tau \lessdot \beta$
- $\alpha, \beta \lessdot t$, being t the present time expressed as absolute.

β in the operator do not indicate optionality. A further explanation of the possible values of α and β can be reviewed in the appendix VI. Examples can be consulted in Section III-B.

Both above introduced operators[3] are not reflexive, hence, they do not include the present. Galton and Augusto [1] introduced an additional operator to express future that remains in the extension: $\oplus$. Where $\oplus[\neg]s$, represents that $[\neg]s$ will happen on the next time unit, unless an external event indicates the opposite.

The syntax for the use of temporal operators in *MTR* is given in the Appendix.

B. Events

Events are used to model impingements on the system from outside. In here intended applications, sensors triggers or human commands. An event is represented as:

$$Occurs(ingr([\neg]s), t^{*})$$

indicating the ingression to *status*, at time t (represented by a natural number $\mathbb{N}$).

Bounded Past Operators in the antecedent can be used to explore the history of any type of states $s \in S_D$ or $s \in S_I$.

C. Rules

The *rule set* R defines two subsets of rules: *same-time rules* R_S and *next-time rules* R_N. Where $R_S \cup R_N = R$ and $R_S \cap R_N = \varnothing$. Same-time rules, are rules that will be applied in the same iteration where the antecedents are satisfied. Next time rules are rules which consequence will be applied on the next iteration than the one where the antecedents are satisfied. *Rules* in R_N, are those with the symbol $\oplus$ in front of the consequence. If the symbol is not present, it will be assumed that the rule is in R_S. The syntax for rules is as follows:

$$[\neg][BP_{Op}][\neg]s_1 \wedge \cdots \wedge [\neg][BP_{Op}][\neg]s_n \rightarrow [\bigoplus][\neg]s$$

[1]Symbol $\dot{-}$ represents a calendar difference function that provides the difference between a calendar date, including clock time, and a value expressed in time units.

[2]Symbols $\leq$ and $\lessdot$, represent a calendar comparison function between two absolute times.

[3]These two operators are the ones used on antecedents of rules. The operator for delayed causation is kept for the consequence.

Where BP_{Op} is either *strong immediate past* or *absolute reference to the past* temporal operator as defied above and $\neg$ is the boolean operator that indicates the status of the *state*. The brackets represent optionality.

Notice we do not allow nested bounded past operators to keep the language simpler and more efficient. Also notice that as in traditional temporal logic where $\Box$ can be linked to $\Diamond$ through $\neg \Diamond \neg p$, we also have the possibility to defined a *stronger* and *weaker* versions of our operators, where by 'stronger' we mean an assertion which reassure us of a state holding uninterruptedly for a period of time and by 'weaker' an assertion which only reassure us that certain condition held for part of a time period. More precise definitions follow.

The negation of the *strong bounded past operators* can lead to the weaker expressions:

1) Weak immediate past, relative to present:

$$\diamondminus_{[\mu]} s = \neg \boxminus_{[\mu]} \neg s$$

Where $\diamondminus_{[\mu]}[\neg]s$, is true if and only if $HoldsAt([\neg]s, \tau)$ for at least one τ where:

- $(t \dot{-} \mu) \leq \tau < t$, being t the present time expressed as absolute.
- $\mu > 0$

An example can be found in Section III-C.

2) Weak absolute reference to the past:

$$\diamondminus_{[\alpha,\beta]} s = \neg \boxminus_{[\alpha,\beta]} \neg s$$

Where $\diamondminus_{[\alpha,\beta]}[\neg]s$, is true if and only if $HoldsAt([\neg]s, \tau)$ for at least one τ where:

- α and β represent absolute time.
- $\alpha \leq \tau \lessdot \beta$
- $\alpha, \beta \lessdot t$, being t the present time expressed as absolute.

An example can be found in section III-D.

D. Stratification

Same-time rules are required to be *stratified*. Stratification eliminates possibilities of loops and structures the algorithm. This is explained as follows.

1) A Stage 1 rule is a rule $s_1 \sqcap s_2 \sqcap \cdots \sqcap s_n \rightarrow s$, where $s_1, \ldots, s_n$ are all independent. In this case s is said to be *1-dependent*. (The independent states are called *0-dependent*.)
2) A Stage k rule is a rule $s_1 \sqcap s_2 \sqcap \cdots \sqcap s_n \rightarrow s$, where each of $s_1, \ldots, s_n$ is at most $(k-1)$-dependent, and at least one of them is $(k-1)$-dependent. Then s is said to be *k-dependent*. In this case we also say that $\neg s$ is *co-k-dependent*.
3) A set of same-time rules is stratified so long as for every rule in the set there is a number k such that the rule is a Stage k rule.

A same-time rule $s_1 \sqcap s_2 \sqcap \cdots \sqcap s_n \rightarrow s$ can be *triggered* at time t so long as we have

$$HoldsAt(s_1, t) \wedge \cdots \wedge HoldsAt(s_n, t).$$

The result of triggering it is that we may assert $HoldsAt(s, t)$. Under the same condition we can also apply a next-time rule $s_1 \sqcap s_2 \sqcap \cdots \sqcap s_n \rightarrow \bigoplus s$, and the result of triggering it is that we may assert $Holds(S, t+1)$. If a rule can be triggered, it is said to be *live*.

E. Forward Reasoning Algorithm

A same-time rule $s_1 \wedge s_2 \wedge \cdots \wedge s_n \rightarrow s$ of stage k is considered *live* at time t (present time), when for any s'_k in the antecedent, either:

1) s'_k is $[\neg]s$ and $HoldsAt([\neg]s, t)$

2) s'_k is $\boxminus_{[\mu]}[\neg]s$ and $HoldsAt([\neg]s, \tau)$ for all possible τ or is $\diamondminus_{[\mu]}[\neg]s$ and $HoldsAt([\neg]s, \tau)$ for at least one τ where:
   - $\mu > 0$
   - $(t \dot{-} \mu) \leq \tau < t$, being t the present time expressed as absolute.

3) s'_k is $\boxminus_{[\alpha,\beta]}[\neg]s$ and $HoldsAt([\neg]s, \tau)$ for all possible τ, or is $\diamondminus_{[\alpha,\beta]}[\neg]s$ and $HoldsAt([\neg]s, \tau)$ for at least one τ where:
   - α and β represent absolute time.
   - $\alpha \leq \tau \lessdot \beta$
   - $\alpha, \beta \lessdot t$, being t the present time expressed as absolute.

Given a stratified set of same-time rules, a set of next-time rules, an *initial condition*, which is specified by determining the truth value of $HoldsAt(s, 0)$ for each $s \in \mathcal{S}_I$, and an *event list*, which is a set of formulae of the form $Occurs(Ingr(s), t^*)$, we compute the resulting history as follows:

1) At $t = 0$, apply any live same-time rules, in order of increasing Stage (note that some rules may become live as a result of applying others). When all possible same-time rules have been applied, apply any live next-time rules[4]. Any positive state s whose value is not already determined at $t = 0$ is now assumed to be false, i.e., $HoldsAt(\neg s, 0)$.
2) For $t = 1, 2, 3, \ldots$,
   a) For each occurrence $Occurs(Ingr(s), (t-1)^*)$, if $Holds(\neg s, t-1)$, assert $HoldsAt(s, t)$.
   b) For each co-independent state s, if $HoldsAt(s, t-1)$ and it has not already been asserted that $HoldsAt(\neg s, t)$, then assert $HoldsAt(s, t)$. This is called 'applying persistence' to the state S.

[4]It is assumed that the specification of a system is not contradictory, i.e., does not represent inconsistent behaviour.

c) For $k = 1, 2, 3, \ldots,$

i) Apply any live same-time rules of Stage k.

ii) Apply persistence: For any co-k-dependent state s, if $HoldsAt(\neg s, t)$ has not already been asserted, and $HoldsAt(s, t - 1)$, then assert $HoldsAt(s, t)$.

d) Apply any live next-time rules.

It has to be noticed that external events have priority over the result of the system in the last iteration.

III. Examples

State persistence will be assumed for all the examples. Additionally, x indicates that the status could be either true or false, because is not relevant for the example.

A. Strong immediate past, relative to present

Table I shows the result of using strong immediate past operator, considering that:

- Is required to know if the state s has been true for the last 3 time units ($\mu = 3$).
- Events:

$$Occurs(ingr(s), 0^*)$$

$$Occurs(ingr(\neg s), 2^*)$$

$$Occurs(ingr(s), 3^*)$$

- Initial settings are $I_c = \{(s, 0)\}$

TABLE I
Expected results for Example III-A

t	s	$\boxminus_{[3]} s$
0	0	0
1	1	0
2	1	0
3	0	0
4	**1**	0
5	**1**	0
6	**1**	0
7	x	**1**

B. Strong Absolute reference to a bounded durative past

Table II shows the result of using strong absolute past operator, realising that:

- It is required to know if the state s was true between at $t = 1$ and $t = 4$.
- $Occurs(ingr(s), 0^*)$

As a special case of the use of this operators, a reference can be done to the minimal duration (instant), where $\alpha = \beta$. The case can be consulted in Table III, taking into account that:

- It is required to know if the state s was true at instant $t = 2$.
- $Occurs(ingr(s), 1^*)$

TABLE II
Expected results for Example III-B

t	s	$\boxminus_{[1,4]} s$
0	x	0
1	**1**	0
2	**1**	0
3	**1**	0
4	**1**	0
5	x	1
6	x	1
7	x	1

TABLE III
Expected results for Example III-B (Instant)

t	s	$\boxminus_{[2,2]} s$
0	x	0
1	x	0
2	1	0
3	x	1
4	x	1

C. Weak immediate past, relative to present

Table IV shows the result of using the weak version of the immediate past operator, noticing that:

- Is required to know if the state s has been true at least one time during the last 3 time units ($\mu = 3$).
- Events:

$$Occurs(ingr(s), 0^*)$$

$$Occurs(ingr(\neg s), 2^*)$$

$$Occurs(ingr(s), 6^*)$$

- Initial settings are $I_c = \{(s, 0)\}$

TABLE IV
Expected results for Example III-C

t	s	$\diamondminus_{[3]} s$
0	0	0
1	1	0
2	1	0
3	0	1
4	0	1
5	0	1
6	0	0
7	1	0
8	x	1

D. Weak Absolute reference to a bounded durative past

Table V shows the result of using weak immediate past operator, assuming:

- It is required to know if the state s has been true at least one time between $t = 1$ and $t = 4$.

- Events:

$$Occurs(ingr(\neg s), 0^*)$$

$$Occurs(ingr(s), 1^*)$$

TABLE V
Expected results for Example III-D

t	s	$\diamondminus_{[1,4]} s$
0	x	0
1	0	0
2	1	0
3	x	0
4	x	0
5	x	1
6	x	1
7	x	1

E. Cooker Unattended Example

It is clear we still keep the possibility to use the full syntax which was allowed in *C* and we kept the algorithm and related notions so we can still process with *M* those scenarios considered in [1]. Our enriched language for the antecedent of rules which are capable to refer to specific portions of the history of the sensor database allows us to deal more effectively with typical scenarios of intelligent environments.

The scenario used in Lu et al. [2] is used to show how the extension can also deal more naturally with durative conditions. Consider the task is to model a kitchen monitored by sensors in a Smart Home System. Assuming the cooker is on ($cookerOn$ represents a sensor detecting cooker being activated), and the presence sensor is not activated ($\neg atKitchen$, $atKitchen$ is a sensor detecting location of a person in the kitchen). If no presence is detected after more than a number of n units of time (it will be assumed that $n = 5$), then, is considered that the cooker is unattended (cu). In this case, the alarm will be on ($alarmOn$) to notify the occupant and the cooker will be switched off ($\neg cookerOn$).

- Independent states: $cookerOn, \pm atKitchen$
- Dependent states: $\pm cu, \pm hazard, \pm alarmOn, \neg cookerOn$
- Same Time Rules:
  - *Stratification Level 1 Rules:*
    $(\boxminus_{[5]} \neg atKitchen \land \boxminus_{[5]} cookerOn \rightarrow cu)$
    $(\neg cookerOn \rightarrow \neg alarmOn)$
    $(\neg cookerOn \rightarrow \neg hazard)$
    $(\neg cookerOn \rightarrow \neg cu)$
  - *Stratification Level 2 Rules:*
    $(cu \rightarrow alarmOn)$
    $(cu \rightarrow hazard)$
- Next Time Rules:
  $(alarmOn \rightarrow \oplus(\neg cookerOn))$
- Events
  $Occurs(ingr(atKitchen), 0:1)$
  $Occurs(ingr(cookerOn), 1:2)$
  $Occurs(ingr(\neg atKitchen), 2:3)$
- Initial Settings:
  $I_c = \{(cookerOn, 0), (atKitchen, 0), (cu, 0), (hazard, 0), (alarmOn, 0)\}$

TABLE VI
Expected results for Example III-E

Time	cookerOn	atKitchen	hazard	alarmOn	cu
0	0	0	0	0	0
1	0	1	0	0	0
2	1	1	0	0	0
3	**1**	**0**	0	0	0
4	**1**	**0**	0	0	0
5	**1**	**0**	0	0	0
6	**1**	**0**	0	0	0
7	**1**	**0**	0	0	0
8	1	0	1	1	1
9	0	0	0	0	0

The improvement here presented, also applies to other examples considered in Lu et al. [6].

IV. Current testing scenario

A. Introduction

Consider the task is to develop a smart office. Two main requirements are presented:

1) **Energy saving:** The system has to regulate the lamp of a desk to avoid energy waste.
2) **Well-being and comfort of the user:**
   a) It will be assumed that the only way for the user to be feed is going to the nearest restaurant. Besides, this restaurant opens only from 1:00 pm to 3:00 pm, and this the only way of getting food for the user.
   b) Daylight hours are considered from 8:00 am to 4:00 pm. Office hours are considered from 9:00 am to 18:00 pm. If user presence is detected out from daylight hours and into office hours, and the lamp is off, the system will automatically switch the lamp on.

B. System Design

The implementation of a system that uses the extension proposed is represented in Fig.1 The system is divided into three main modules: *Log Reader*, *Algorithm* and *Actuator Manager*. All of them are written in *Java programming language*. Additionally, there is a *sensor*, a couple of *actuators*, a *database*, a *file* containing the definition of the system rules and a *router*.

1) Sensor (S_1): A *HomePro ZIR010 RF Transmitter PIR* motion sensor that sends sensed information about any motion activity detected to the router periodically using a *Z-Wave wireless communication protocol* (a).

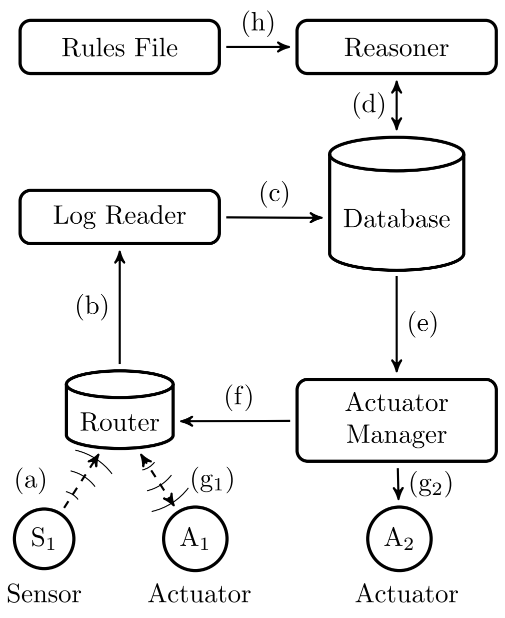


Fig. 1. Global structure of the implemented system.

2) Router: A *Vera Router* manufactured by *Mi Casa Verde*, which provides a framework to control devices working with *Z-Wave*.
3) Log Reader: The main function of this module is to read the log that the router is continuously updating using Secure Shell (SSH) network protocol to retrieve the latest information related with the sensors (b). Besides, it saves the relevant data read from the log into the database (c).
4) Database: The database used is a PostgreSQL database that has four tables. Events from the router, previously interpreted by the log reader module, are inserted into *Events table* (Table VII) (c). The algorithm module retrieves information from that table, relating devices to states with *Sensors table* (Table VIII) (d). An *Internal Events table* (Table IX) is used to handle internal events that are triggered every time a next time rule becomes live (d). Finally, the algorithms saves each iteration into the *Results table* (Table X) (d). Information from bounded past operators is also retrieved from it.
5) Reasoner: This module applies the *Forward Reasoning Algorithm* explained in Section II-E, with the time bounded operators. Before the first step (Section II-E: 1), it reads the rules and states of the system from a *rules file* (h). For the step Section II-E: 2a, latest changes from *Events table* (Table VII) are read. Finally, after doing the last step (Section II-E: 2d), the status of the system in that precise iteration is saved into the *Results table* (Table X).
6) Actuator Manager: This module is used to read data values from the database continuously (e) and reflect changes into actuators:
   a) Lamp (A_1): A lamp connected to an *Everspring AN148-3* On/Off switch. Communication between the manager and the router (f, g_1) is made using Hypertext Transfer Protocol to access the Web Services that the router provides for the switch (f). Finally, the router uses *Z-Wave* to send the commands received through the services (g_1). Additionally, periodic information of its status is sent (g_1).
   b) Screen(A_2): Based on a *JFrame Window*, shows an advise to the user in a screen. *Actuator Manager* can open and close the window (g_2).
7) Rules file: It is a *Microsoft Windows Text File (txt)* with the initial configuration of the system, the rules and the states (including bounded past operators); expressed in *American Standard Code for Information Interchange (ASCII)*. (See Fig. 2).

TABLE VII
EVENTS TABLE FROM THE DATABASE.

id	PK	Serial
device	FK	varchar('50')
value		boolean
date		date
time		time

TABLE VIII
SENSORS TABLE FROM THE DATABASE.

device	PK	varchar('50')
state		varchar('50')

TABLE IX
INTERNAL EVENTS TABLE FROM THE DATABASE.

id	PK	serial
state		varchar('50')
value		boolean

TABLE X
RESULTS TABLE FROM THE DATABASE.

$System\_time\_millis$	PK	bigint
s_1		boolean
s_2		boolean
$\vdots$		$\vdots$
s_n		boolean

C. System rules

1) **Energy saving:** It will be assumed that the lamp is on (lampOn represents the lamp actuator sensor detecting that the lamp in the desk is on (A_1 from Figure 1). If the motion sensor is not activated ($\neg atDesk$) after more than a number of n units of time (It will be assumed

$n = 30$ time units), then the system will switch the lamp off ($\neg lampOn$).

2) **Health and comfort of the user:**
   a) If the system detects presence ($atDesk$) for all the instants between 12:45 pm and 2:30 pm, it will assume that the user has not eat, and it will display a reminder message ($advise$). The advise will be removed ($\neg avise$) 1 minute after displaying. This advise will not appear again until the next day.
   b) If user presence ($atDesk$) is detected out from 4:00 pm to 6:00 pm, and the lamp is off ($\neg lampOn$), the system will switch on the lam ($lampOn$).

- Independent states: $\pm atDesk$
- Dependent states: $\pm advise, \pm adviseEnabled, \pm lampOn$
- Same Time Rules:
  - *Stratification Level 1 Rules:*
    $(\diamondminus_{[60]} \neg advise \rightarrow \neg adviseEnabled)$
  - *Stratification Level 2 Rules:*
    $(\boxminus_{[23:59:59,23:59:59]} \neg adviseEnabled \rightarrow adviseEnabled)$
- Next Time Rules:
  $(\diamondminus_{[12:30,14:30]} atDesk \wedge adviseEnabled \rightarrow \oplus advise)$
  $(\diamondminus_{[16:00,18:00]} atDesk \wedge \neg lampOn \rightarrow \oplus lampOn)$
  $(\boxminus_{[30]} \neg atDesk \wedge lampOn \rightarrow \oplus \neg lampOn)$
- Events
- Initial Settings:
  $I_c = \{(atDesk, 0), (\neg advise, 0),$
  $(\neg adviseEnabled, 0), (lampOn, 0)\}$

D. System Implementation

The results of the case explained in Section IV-A: 1 are shown in Fig. 3. Letters between brackets correspond to Fig.1, Section IV-B letters.

1) The image from Fig. 3.I, is the *Log Reader*, interpreting the log of the router (b). It can be seen that the *device 3* has value 0 (false).
2) Fig. 3.II, shows the values saved into the database (c). It can be noticed that in the database, the relation with the *device number 3* and the state $atDesk$ is done. The time from columns *date* and *time* is the time from the *router* log. The column *timestamp* saves the value of the time when the *Log Reader* module detects the ingression of the event. There is a time difference between columns, this is because the *router* takes the time from the controller box and the *Log Reader* module from the computer is running in. The query that the image shows, is done against two database tables: *Events Table* and *Sensors Table*. (Table VII and Table VIII respectively, from section IV-B)
3) Fig. 3.III, shows what happens inside the *Reasoner* module. Three different parts can be seen:
   - First, the module detects an event into the system (d).
   - Then, after 30 units of time, triggers the next time rule to switch off the lamp. The consequence is sent to the database as an event (d) (Fig. 3.IV).
   - On the following iteration, this event is read from the database. Concretely, from the *Internal Events Table* (Table IX, from section IV-B)
4) Fig. 3.V, shows that the *Results table* (Table X, from section IV-B), that contains all the status of the states for each time unit (d).
5) Finally in 3.VI, the *Actuator Manager* module calls the service in the router to switch off the light (f).

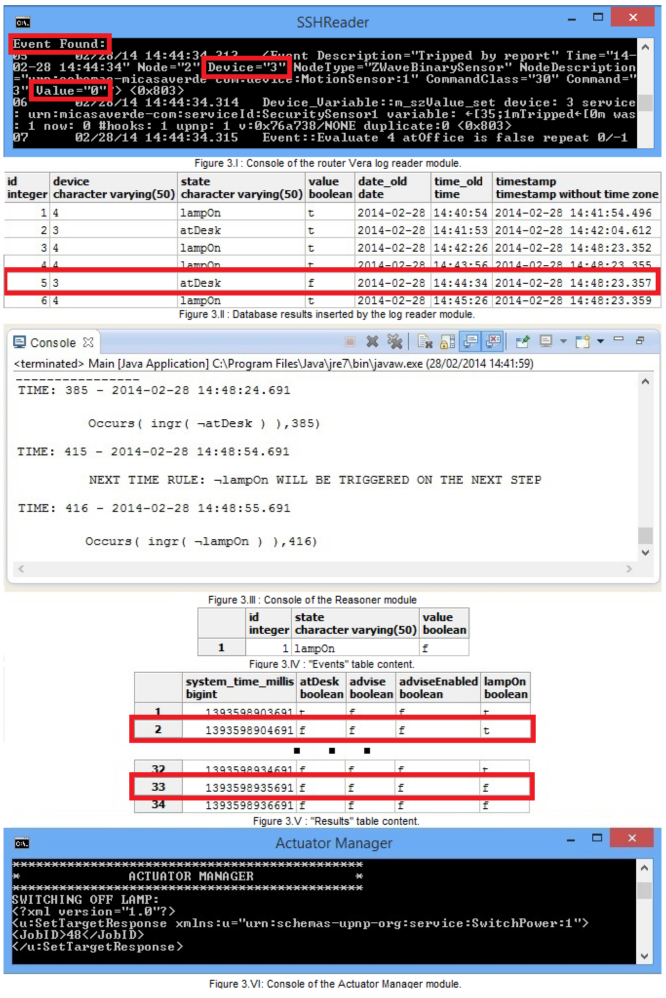
```
states([atDesk,advise,adviseEnabled,lampOn]).

is(atDesk).
is(#atDesk).

holdsAt(atDesk,0).
holdsAt(lampOn,0).
holdsAt(#advise,0).
holdsAt(#adviseEnabled,0).

ssr( (<->[60]#advise) => #adviseEnabled ).
ssr( ([-][{_._._.23.59.59}-{_._._.23.59.59}]#adviseEnabled) => adviseEnabled).
ssr( (<->[{_._._.12.30.00}-{_._._.14.30.00}]atDesk) => advise).
ssr( (<->[{_._._.16.00.00}-{_._._.18.00.00}]atDesk,#lampOn) => lampOn).
sEr( ([-][30]#atDesk,lampOn) => #lampOn).
```

Fig. 2. Rules File for the office example explained in Section IV-A.



SSHReader

```
Event Found:
05      02/28/14 14:44:34.313   <Event Description="Tripped by report" Time="14-
02-28 14:44:34" Node="2" Device="3" NodeType="ZWaveBinarySensor" NodeDescription
="urn:schemas-micasaverde-com:device:MotionSensor:1" CommandClass="30" Command="
3" Value="0" > <0x803>
06      02/28/14 14:44:34.314   Device_Variable::m_szValue_set device: 3 service
: urn:micasaverde-com:serviceId:SecuritySensor1 variable: ←[35;1mTripped←[0m was
: 1 now: 0 #hooks: 1 upnp: 1 v:0x76a738/NONE duplicate:0 <0x803>
07      02/28/14 14:44:34.315   Event::Evaluate 4 atOffice is false repeat 0/-1
```

Figure 3.I : Console of the router Vera log reader module.

id integer	device character varying(50)	state character varying(50)	value boolean	date_old date	time_old time	timestamp timestamp without time zone
1	4	lampOn	t	2014-02-28	14:40:54	2014-02-28 14:41:54.496
2	3	atDesk	t	2014-02-28	14:41:53	2014-02-28 14:42:04.612
3	4	lampOn	t	2014-02-28	14:42:26	2014-02-28 14:48:23.352
4	4	lampOn	t	2014-02-28	14:43:56	2014-02-28 14:48:23.355
5	3	atDesk	f	2014-02-28	14:44:34	2014-02-28 14:48:23.357
6	4	lampOn	t	2014-02-28	14:45:26	2014-02-28 14:48:23.359

Figure 3.II : Database results inserted by the log reader module.

Console

```
<terminated> Main [Java Application] C:\Program Files\Java\jre7\bin\javaw.exe (28/02/2014 14:41:59)
----------------
TIME: 385 - 2014-02-28 14:48:24.691

        Occurs( ingr( ¬atDesk ) ),385)

TIME: 415 - 2014-02-28 14:48:54.691

        NEXT TIME RULE: ¬lampOn WILL BE TRIGGERED ON THE NEXT STEP

TIME: 416 - 2014-02-28 14:48:55.691

        Occurs( ingr( ¬lampOn ) ),416)
```

Figure 3.III : Console of the Reasoner module

	id integer	state character varying(50)	value boolean
1	1	lampOn	f

Figure 3.IV : "Events" table content.

	system_time_millis bigint	atDesk boolean	advise boolean	adviseEnabled boolean	lampOn boolean
1	1393598903691	t	f	f	t
2	1393598904691	f	f	f	t
■ ■ ■					
32	1393598934691	f	f	f	t
33	1393598935691	f	f	f	f
34	1393598936691	f	f	f	f

Figure 3.V : "Results" table content.

Actuator Manager

```
*********************************************************
*                 ACTUATOR MANAGER                      *
*********************************************************
SWITCHING OFF LAMP:
<?xml version="1.0"?>
<u:SetTargetResponse xmlns:u="urn:schemas-upnp-org:service:SwitchPower:1">
<JobID>48</JobID>
</u:SetTargetResponse>
```

Figure 3.VI: Console of the Actuator Manager module.

Fig. 3. System results.

V. Conclusions and Future Work

This paper introduced a system, M, which includes a language to define contexts of interest which is based on the natural characteristics of reactive intelligent environments capable to track certain environmental conditions and to act upon those. We have explained the language in detail, especially the capabilities to intuitively capture previous important states happening at a particular instant or interval of time by using metric temporal operators. We kept the simple and elegant algorithm of a previous system and also the rule based nature.

We have illustrated how the language can be used to intuitively capture common situations and we have also explained our current testing framework which takes real data from sensors in an office and provides actuation in real-time. We explained the technological infrastructure of the system and illustrated its use with a smart office example.

The structure of the rules in M are similar to those produced as an output by *LFPUBS* which is a system capable to learn user behaviour from intelligent environments. Our next step is to connect these two systems (M and *LFPUBS*) in a closed loop so that the reasoner can benefit from the learnt behaviours.

Acknowledgment

The first author wishes to acknowledge the help from Mondragon University and the Basque Country for the funding of his stay at Middlesex University in London.

References

[1] A. Galton and J. C. Augusto, "Stratified causal theories for reasoning about deterministic devices and protocols," in *TIME*, 2002, pp. 52–54.

[2] Z. Lu, J. Augusto, J. Liu, H. Wang, and A. Aztiria, "A system to reason about uncertain and dynamic environments," *International Journal on Artificial Intelligence Tools*, vol. 21, no. 05, 2012.

[3] A. Aztiria, J. C. Augusto, R. Basagoiti, A. Izaguirre, and D. J. Cook, "Learning frequent behaviors of the users in intelligent environments," *IEEE T. Systems, Man, and Cybernetics: Systems*, vol. 43, no. 6, pp. 1265–1278, 2013.

[4] J. C. Augusto, V. Callaghan, D. Cook, A. Kameas, and I. Satoh, "Intelligent environments: a manifesto," *Human-centric Computing and Information Sciences*, vol. 3, no. 1, pp. 1–18, 2013.

[5] J. C. Augusto, "The logical approach to temporal reasoning," *Artificial Intelligence Review*, vol. 16, no. 4, pp. 301–333, 2001.

[6] Z. Lu, J. Augusto, J. Liu, and H. Wang, "A linguistic truth-value temporal reasoning (ltr) system and its application to the design of an intelligent environment," *International Journal of Computational Intelligence Systems*, vol. 5, no. 1, pp. 173–196, 2012.

VI. Appendix

```
TemporalOperators ::=
ImmediatePast | AbsolutePastReference

        ImmediatePast ::=
        StrongIP | WeakIP

                StrongIP ::=
                '[-]' '[' Number ']' State

                WeakIP ::=
                '<->' '[' Number ']' State

        AbsolutePastReference ::=
        StrongAPR  | WeakAPR

                StrongAPR  ::=
                '[-]'  Interval State

                WeakAPR  ::=
                '<->'  Interval State

Interval  ::=
'[' AbsoluteTime , AbsoluteTime ']'

AbsoluteTime ::= '{' Date '.' Time '}'

Date ::= Year '.' Month '.' Day

Year ::=
YearNumber | UndefinedParameter

        YearNumber ::=
        ( a four digit natural number )

Month ::=
'january' | 'february' | ...
| 'december' | UndefinedParameter

Day ::=
'1' | '2' | ...
| '31' | UndefinedParameter

Time ::= Hour '.' Minute '.' Second

Hour ::=
'0' | '1' | ...
| '23' | UndefinedParameter

Minute ::=
'0' | '1' | ...
| '59' | UndefinedParameter

Second ::=
'0' | '1' | ...
| '59' | UndefinedParameter

UndefinedParameter ::= '_'
Number ::= (a natural number)
YearNumber ::= ( a four digit natural number )
State ::= (any state of the system)
```