Event Calculus agent minds applied to diabetes monitoring

Nicola Falcionelli[1], Paolo Sernani[1], Albert Brugués[2], Dagmawi Neway Mekuria[1], Davide Calvaresi[2,3], Michael Schumacher[2], Aldo Franco Dragoni[1], Stefano Bromuri[4]

[1] Università Politecnica delle Marche, Ancona, Italy
{n.falcionelli, d.n.mekuria}@pm.univpm.it,
{p.sernani, a.f.dragoni}@univpm.it

[2] University of Applied Sciences Western Switzerland, Sierre, Switzerland
{albert.brugues, michael.schumacher}@hevs.ch

[3] Scuola Superiore Sant'Anna, Pisa, Italy
d.calvaresi@sssup.it

[4] Open University of the Netherlands, Heerlen, the Netherlands
stefano.bromuri@ou.nl

Abstract. The increasing incidence of chronic diseases is a major challenge for the healthcare sector. Personal Health Systems (PHSs) address the self-management of chronic diseases, by decentralizing the health monitoring outside hospitalized environments. Rule based agents allow bringing domain experts' knowledge into PHSs. However, agents must meet the requirements of real monitoring scenarios, characterized by massive streams of events. Hence, with the aim to monitor the health status of diabetic patients, two logic-based agent minds for an agent-oriented PHS are presented. One agent mind is based on the standard version of jREC, a Prolog-based implementation of Cached Event Calculus, while the other is a customization of the standard jREC mind that exploits an event-indexing technique. Both of them are as well integrated into MAGPIE, a Java agent platform. The paper then compares and analyzes the performances of the proposed agent minds, by computing the time needed to trigger different type of alerts, when the number of recorded events (e.g. values of physiological parameters) increases. The results show that the customized jREC mind performs much better when an high number of events need to be checked, making its use advisable in monitoring scenarios.

1 Introduction

The incidence of chronic diseases in the population is recognized as a major challenge for the healthcare sector [2]. For instance, the number of people affected by diabetes has doubled in the last 20 years [33]. Statistics from WHO report that more than 400 million individuals live with diabetes, and losses in the GDP for diabetes-related costs from 2011 to 2030 are estimated at 1.7 trillion USD [32].

Personal Health Systems (PHSs) aim at supporting the self-management of chronic diseases and reducing the healthcare costs, supporting medical doctors in following the patients' disease evolution [10]. PHSs implement the "*healthcare to anyone, anytime, and anywhere*" paradigm, by increasing both the coverage and the quality of healthcare [31]. In fact, PHSs bring the health technology to domestic environments, by localizing healthcare services to the specific needs, practices, and situations of people and their social contexts [24]. PHSs ensure the continuity of care, focusing on a knowledge-based approach integrating past and current data of each patient together with statistical evidence [29]. A PHS is composed of three tiers [30]: Tier 1 is the Body Area Network (BAN), i.e. the set of sensors on the patient's body to monitor her health parameters; Tier 2 is the personal server, usually a mobile device, which collects and aggregates the parameters and events produced by the BAN; Tier 3 is the remote server which processes and stores the data from the personal server and supports doctors in following the treatment of patients at home.

Beyond the modeling capabilities of agent-based frameworks [28] and their still opened challenges [13, 12], Multi-Agent Systems have been proved useful in the healthcare sector implementing modularity, distribution, and personalization for data management, decision support systems, planning and resource allocation, and remote care [18], being ideal for PHSs. In [8], an agent-based platform called MAGPIE implements a programmable expert PHS to monitor patients suffering from diabetes. In particular, that agent platform adds scalability to the PHS by shifting from Tier-3 to Tier-2 the computation needed for the patient monitoring. To obtain such scalability, the agents, composed by an agent body and an agent mind, run directly on the personal server. The agent body is the part of the agent that collects the data acting as an interface between the BAN in Tier-1 and the agent mind. The agent mind, based on an Event Calculus (EC) engine, is the part of the agent that checks the data collected from the body to perform the monitoring task and trigger alerts for the medical doctors logging in Tier-3. The approach of MAGPIE allows improving the scalability of the PHS when the number of patients increases, compared to a centralized PHS where the computation is performed in Tier-3. However, another aspect has to be taken into account: the scalability of the agent mind when the number of events increases. In fact, the use of rule engines based on EC usually restricts the number of events and rules to be applied in a real monitoring scenario, where short time delays are needed to apply corrective actions. Thus, the next step to apply the agent-based PHS in real scenarios requiring long-term monitoring is to develop agent minds capable of caching and retrieving events efficiently.

This paper addresses such issue by proposing two agent minds for the MAGPIE agent platform presented in [8]. The agent minds have been implemented using jREC, a Cached Event Calculus (CEC) reasoner based on Java and tuProlog [5], to move the computational complexity from query to update time by caching the maximum validity intervals for fluents. Even if both based on jREC and integrated into the MAGPIE agent platform, the proposed agent minds differ on the way in which they handle event streams. One is a straightforward

integration of the jREC engine, and the other is based on an indexing technique that gives to jREC the ability to process event streams more efficiently.

In addition, as the main contribution of the paper, the performances of the jREC-based agent minds are evaluated on the time required to trigger an alert, when the number of events generated by the agent body increases. Diabetes has been adopted as the use case for the monitoring rules to be checked.

The rest of the paper is organized as follows. Section 2 presents the paper background on EC, CEC, jREC and red-black trees. Section 3 describes an overview of the entire PHS in which the agent minds runs, shows the encoding of the monitoring rules to check alerts based on glucose and blood pressure levels in diabetic patients, as well as problems and solutions that arise when massive streams of events have to be handled by reasoning engines. Section 4 presents the experimental results to evaluate the two agent minds. Section 5 describes the work related to the presented research. Section 6 draws the conclusions of the paper and outlines the future work.

2 Background

This section introduces the concepts on which the proposed agent minds are based on.

2.1 Event Calculus

EC is a logic formalism for reasoning about actions and their effects in time [20]. Therefore, it is a suitable tool for modeling expert systems representing the evolution in time of an entity by means of the production of events. EC is based on many-sorted first-order predicate calculus, known as domain-independent axioms, which are represented as normal logic programs that are executable in Prolog. The underlying time model of EC is linear. EC manipulates fluents, where a fluent represents a property that can have different values over time. The term F=V denotes that a fluent F has value V as a consequence of an action that took place at some earlier time-point and not terminated by another action in the meantime. Table 1 summarizes the main EC predicates. Predicates, functions, symbols and constants start with lowercase letter, while variables start with uppercase letter. Predicates in the text are referenced as predicate/N, where predicate is the name of the predicate and N its arity (e.g. number of arguments).

The domain independent axioms of EC are the following:

$$\text{holdsAt}(F = V, 0) \leftarrow \text{initially}(F = V). \tag{1}$$

$$\begin{aligned} &\text{holdsAt}(F = V, T) \leftarrow \\ &\qquad \text{initiatesAt}(F = V, T_s), T_s < T, \\ &\qquad \text{not broken}(F = V, [T_s, T]). \end{aligned} \tag{2}$$

Predicate (1) states that a fluent F holds value V at time 0, if it has been initially set to this value. For any other time T $>$ 0, the predicate (2) states that

Table 1: Main Event Calculus predicates

Predicate	**Meaning**
initially(F=V)	The value of fluent F is V at time 0
holdsAt(F=V,T)	The value of fluent F is V at time T
holdsFor(F=V,[T_{min},T_{max}])	The value of fluent F is V between T_{min} and T_{max}
initiatesAt(F=V,T)	At time T the fluent F is initiated to have value V
terminatesAt(F=V,T)	At time T the fluent F is terminated from having value V
broken(F=V,[T_{min},T_{max}])	The value of fluent F is either terminated at T_{max}, or initiated to a different value than V between T_{min} and T_{max}
happensAt(E,T)	An event E takes place at time T updating the state of the fluents

the fluent holds at time T if it has been initiated to value V at some earlier time point Ts, and it has not been broken on the meanwhile.

$$\begin{aligned} &\text{broken}(F = V, [Tmin, Tmax]) \leftarrow \\ &\quad \text{terminatesAt}(F = V, T), Tmin < T, Tmax > T. \end{aligned} \tag{3}$$

$$\begin{aligned} &\text{broken}(F = V_1, [Tmin, Tmax]) \leftarrow \\ &\quad \text{initiatesAt}(F = V_2, T_i), V_1 \neq V_2, \\ &\quad Tmin < Ti, Tmax > Ti. \end{aligned} \tag{4}$$

Predicates (3) and (4) specify the conditions that break a fluent. Predicate (3) states that a fluent is broken between two time points Tmin and Tmax if within this interval it has been terminated to have value V. Alternatively, predicate (4) states that a fluent is broken within a time interval if it has been initiated to hold a different value.

$$\begin{aligned} &\text{holdsFor}(F = V, [Tmin, Tmax]) \leftarrow \\ &\quad \text{initiatesAt}(F = V, Tmin), \\ &\quad \text{terminiatesAt}(F = V, Tmax), \\ &\quad \text{not broken}(F = V, [Tmin, Tmax]). \end{aligned} \tag{5}$$

$$\begin{aligned} &\text{holdsFor}(F = V, [Tmin, infPlus]) \leftarrow \\ &\quad \text{initiatesAt}(F = V, Tmin), \\ &\quad \text{not broken}(F = V, [Tmin, +\infty]). \end{aligned} \tag{6}$$

$$\begin{aligned} \text{holdsFor}(F = V, [infMin, Tmax]) \leftarrow \\ \text{terminatesAt}(F = V, Tmax), \\ \text{not broken}(F = V, [-\infty, Tmax]). \end{aligned} \tag{7}$$

Predicates (5), (6) and (7) deal with the validity intervals of fluents. In particular, predicate (5) specifies that a fluent F keeps value V for a time interval going from Tmin to Tmax if nothing happens in the middle that breaks such an interval. Predicates (6) and (7) behave in the same way, but deal with open intervals.

The domain dependent predicates in EC are typically expressed in terms of the initiatesAt/2 and terminatesAt/2 predicates. One example of a common rule for initiatesAt/2 is

$$\begin{aligned} \text{initatesAt}(F = V, T) \leftarrow \\ \text{happensAt}(Ev, T), \\ Conditions[T]. \end{aligned} \tag{8}$$

The above definition states that a fluent is initiated to value V at time T if an event Ev happens at this time point, and some optional conditions depending on the domain are satisfied. In relation with MAGPIE, the agent platform in which the proposed agent mind has been integrated, these events that must happen are physiological measurements from the patient.

2.2 Cached Event Calculus and jREC

Straightforward implementations of EC [20] have time and memory complexity which are not practical for developing real applications. This is due to the fact that every time the EC engine is queried, the computation starts from scratch, and all fluents validity intervals are calculated again. Cached Event Calculus (CEC), proposed by Chittaro and Montanari [14], tries instead to overcome this inefficiency by giving EC a memory mechanism, and moving computation from query time to update time.

CEC formalizes the concept of Maximal Validity Interval (MVI), that represents a time interval in which a particular fluent holds without being terminated by any event. A fluent is also associated to a list of MVIs, in order to express all the time intervals in which that fluent holds continuously.

Whenever the rule engine is updated (e.g. by inserting a new event occurrence), the fluents' MVIs are calculated, and then stored for further use, allowing incremental computation for following updates. Also, every time a new event is added to the database, CEC manages to compute MVIs only for the fluents that can vary with that event, and does not check the MVIs of those fluents that cannot possibly change, thus avoiding unnecessary computation.

jREC is a reasoning tool based on Java and tuProlog that implements a lightweight version of CEC [5]. Since MAGPIE is also written in Java, it has been chosen to implement the proposed agent minds, in order to ensure seamless integration with the agent platform.

jREC consists of three main components:

– The Prolog theory, which represents the actual CEC axiomatization that is loaded into tuProlog;
– The Java engine, which allows to query and update the database without having to interact directly with tuProlog, as well as adding specific domain-dependent theories;
– The Tester, which is a GUI based stand-alone tool for editing theories, visualizing fluents' MVIs and event occurrences, mainly used for prototyping and developing domain-dependent theories.

2.3 Red-black trees

A red-black tree (RBT) is a well known data structure proposed by Rudolf Bayer in 1972 [3]. It is a binary search tree which provides $O(log(n))$ Worst Case time complexity for operations such as node searching, insertion and deletion, as well as $O(n)$ Worst Case space complexity [3]. This is made possible thanks to node coloration: every node of the tree is augmented with an extra bit, and based on the value of such bit, the node is considered to be red or black.

The aforementioned operations rely on such coloration feature to achieve Worst Case logarithmic time complexity and linear space complexity. In fact, every operation that modifies the RBT has to comply with very precise policies which constrain how the nodes should be moved or re-painted. These policies guarantees that the nodes in an RBT are always balanced after every operation, giving such data structure the epiteth of self-balancing. Even though the obtained balance is not perfect, it is proven to be good enough to provide the declared performances [3].

Red-black trees can be effectively exploited as indexing data structures. As it will be also explained in Section 3, one of the agent minds that are proposed in this work relies on such RBT-based indexing in order to efficiently process event streams.

3 System overview

The implemented agent minds run in Tier-2 of the MAGPIE agent-based PHS for self monitoring of diabetes. The entire PHS is depicted in Figure 1. Each patient has its own agent composed by a body (Tier-1) and a mind (Tier-2) running on the personal server: in Tier-1 data are collected from the patients through a BAN; in Tier-2, the agent minds are responsible to trigger possible alerts based on the patients' physiological values, running domain dependent rules which could be customized for each patient. The triggered alerts have to be sent as a notification to medical doctors connected to Tier-3.

3.1 MAGPIE Agent Platform

MAGPIE is an agent platform integrated with the Android OS. It plays the role of Tier-2 in a PHS by connecting the patient and the medical doctor, with

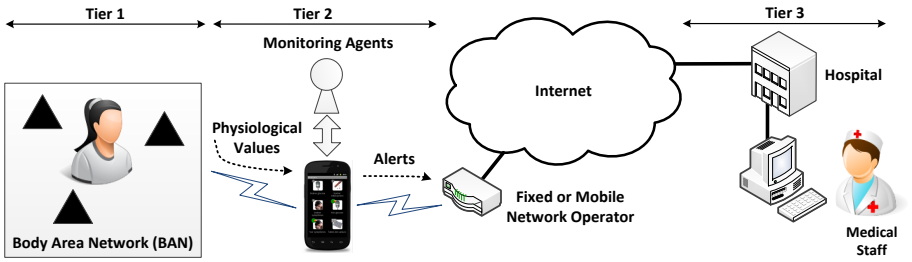


Fig. 1: The agentified PHS. The agent mind runs in Tier-2, to monitor the patient's physiological values.

the aim of improving the management of chronic diseases. From the side of the patient it collects physiological values, whereas from the medical side it models the medical knowledge in terms of monitoring rules expressed as domain dependent axioms of EC. Interested readers can find in [9] a description of the MAGPIE architecture and its integration with Android. In relation to this work, a monitoring rule is defined as a combination of events that trigger an alert to be notified to a medical doctor, where an event is considered as the measurement of a physiological parameter. Therefore, the following two types of monitoring rules are specified:

- Complex rules: consist of the combination of two or more events in a specific time window, where the order in which the events happen is not considered.
- Sequential rules: consist of the sequence of two or more events in a specific time window, where the particular order in which the events occur matters.

3.2 Diabetes Monitoring Rules

In order to detect alert conditions related to diabetes, a sequential and a complex rule patterns are proposed. These rule patterns are based on the literature available for glucose and blood pressure monitoring [16, 6] and checks physiological values collected by the patient's BAN. The patterns identify alert conditions in the patient's health status by modeling the sensor inputs as events that are evaluated in the body of the rules. The two patterns are:

Pattern 1: Brittle diabetes, defined as a glucose rebound going from less than 3.8 mmol/l to more than 8.0 mmol/l in a period of six hours. This pattern can be expressed with a sequential rule.

Pattern 2: Pre-hypertension, defined as two events of high blood pressure in a period of one week. This pattern can be expressed with a complex rule.

Pattern 1 is implemented as follows:

$$\begin{aligned}&\text{initiatesAt}(F = A, T) :- \\ &\qquad \text{happensAt}(\text{ev}(2, A, W), T), \\ &\qquad \text{happensAt}(\text{ev}(1, A, \_), T_1), \\ &\qquad T_s \text{ is } (T - W), \\ &\qquad T > T_1, \\ &\qquad T_1 >= T_s, \\ &\qquad \text{no\_alert}(A, T_s).\end{aligned} \tag{9a}$$

$$\begin{aligned}&\text{terminatesAt}(F = A, T) :- \\ &\qquad \text{happensAt}(\text{ev}(1, A, \_), T).\end{aligned} \tag{9b}$$

$$\begin{aligned}&\text{happensAt}(\text{ev}(1, \text{`brittle diabetes'}, E), T) :- \\ &\qquad \text{hours\_to\_epoch}(6, E), \\ &\qquad \text{happensAt}(\text{glucose}(G), T), \\ &\qquad G =< 3.8.\end{aligned} \tag{9c}$$

$$\begin{aligned}&\text{happensAt}(\text{ev}(2, \text{`brittle diabetes'}, E), T) :- \\ &\qquad \text{hours\_to\_epoch}(6, E), \\ &\qquad \text{happensAt}(\text{glucose}(G), T), \\ &\qquad G >= 8.\end{aligned} \tag{9d}$$

Rules (9a) and (9b) represent a generic sequential rule template with two events. In particular, the fluent F (i.e. the alert) is initiated with value A when: (i) two temporal ordered events occur inside a certain time window and (ii) when the fluent does not hold anywhere else inside the time window (no_alert/2). The fluent F is instead terminated when the first event of the ordering happens.

Rules (9c) and (9d) customize the template for the glucose monitoring use case. They instantiate the variables of the ev/3 term, specifying the time window width (W), the alert name (A) and the threshold values for G.

Pattern 2 is expressed in the following way:

$$\begin{aligned}&\text{initiatesAt}(F = A, T) :- \\ &\qquad \text{happensAt}(\text{alertcheck}(A, W, NMax_1), T), \\ &\qquad T_s \text{ is } (T - W), \\ &\qquad \text{count\_events\_tw}(N_1, \text{evc}(1, A), T_s, T), \\ &\qquad N_1 >= NMax_1, \\ &\qquad \text{no\_alert}(A, T_s).\end{aligned} \tag{10a}$$

$$\begin{aligned}&\text{terminatesAt}(F = A, T) :- \\ &\qquad \text{happensAt}(\text{alertcheck}(A, W, \_), T), \\ &\qquad \text{holdsAt}(F = A, T).\end{aligned} \tag{10b}$$

$$\begin{aligned}\text{happensAt}(\text{evc}(1, \text{`pre-hypertension'}), T) :- \\ \text{happensAt}(\text{blood\_pressure}(S, D), T), \\ S >= 130, \\ D >= 80.\end{aligned} \tag{10c}$$

$$\begin{aligned}\text{happensAt}(\text{alertcheck}(\text{`pre-hypertension'}, E, 2), T) :- \\ \text{weeks\_to\_epoch}(1, E), \\ \text{happensAt}(\text{evc}(1, \text{`pre-hypertension'}), T).\end{aligned} \tag{10d}$$

Rules (10a) and (10b) represent a generic complex rule template with one event type. In particular, the fluent F (i.e. the alert) is initiated with value A when: (i) there are least $NMax_1$ occurrences of the alertcheck/3 event inside the time window and (ii) when the fluent does not hold anywhere else inside the time window (no_alert/2). Also, the count_events_tw/4 predicate is necessary to handle different event temporal orderings without having to duplicate the rule body for every permutation. Rules (10c) and (10d) customize the template for the hypertension monitoring use case. They instantiate the variables of the evc/2 and the alertcheck/3 terms specifying the time window width (W), the alert name (A) and the threshold values for S and D.

3.3 Event Handling with jREC

Efficient handling of massive event streams, while preserving the philosophy of Event Calculus, and in broader terms, of Logic Programming, is a non-trivial task. Techniques such as (i) event windowing/forgetting [1], (ii) theory pre-compilation [1] and (iii) a priori assumptions on event temporal ordering, can help to ease the burden of this process, but at the same time their adoption will cause the reasoning approach to be less general and less flexible. Therefore, since in real case monitoring scenarios these techniques and assumptions might simply not be applicable, finding alternatives ways to tackle the problem in a more general case becomes mandatory.

For example, jREC does not apply any simplifying assumption or technique to the event streams: this forces the reasoner to spend a very high amount of resources every time the engine's knowledge base (KB) is updated with new events. Whenever a list of new events has to be asserted into the KB, jREC must perform the following steps:

- Sort the list of new events chronologically;
- Read all the events already present in the KB and put them in a list;
- Retract all the events from the KB;
- Sort the list of KB's events chronologically;
- Merge the list of new events with the list of events read from the KB;
- Sort the newly obtained list chronologically and remove duplicates;
- Assert the events from the newly obtained list back into the KB;
- Calculate the effects on the fluents.

This procedure indeed maintains the reasoning as general and flexible as possible, but it is also the main source of jREC inefficiency, since every new event(s) insertion causes the engine to sort the event lists multiple times.

To tackle such issue, this paper proposes the integration of jREC with an indexing data structure, i.e. the previously mentioned red-black trees. RBTs will take the duty of maintaining the events temporal ordering by avoiding unnecessary sorting operations, and ensuring fast execution times.

The introduction of event indexing with RBTs allows to more precisely define the proposed agent minds:

- An agent mind based on the standard jREC implementation (standard jREC);
- An agent mind based on a custom jREC implementation, which has been augmented with RBT event indexing (RBT-index jREC).

It should be noticed that, since (i) an event normally contains multi-dimensional data (i.e. timestamp and phyisiological values), (ii) an RBT only allows single-dimensional indexing, and (iii) jREC needs the events to be ordered chronologically, the only choice is to consider the events timestamp as the key on which the indexing will be performed.

4 Test setup and results

The performances of the two jREC agent minds have been evaluated using the sequential and complex rule patterns described above. To accomplish that, synthetic datasets containing glucose and blood pressure measurements have been created. Each measurement is a tuple containing the value(s) and its timestamp.

4.1 Testing Protocol

To see how the performances of the agent minds evolve when the number of events increases, a series of random dataset has been created, each one containing a different number of events.

The events of each dataset are fed into the agent minds one by one, and the time needed by each agent to trigger the alert is recorded. Every experiment is repeated one-hundred times to obtain the mean and standard deviation values.

The biggest assumption of the experiment is that real datasets do not add any value to the performance evaluation. In fact, the use of synthetic datasets allowed to stress the agent minds on very specific and critical tasks.

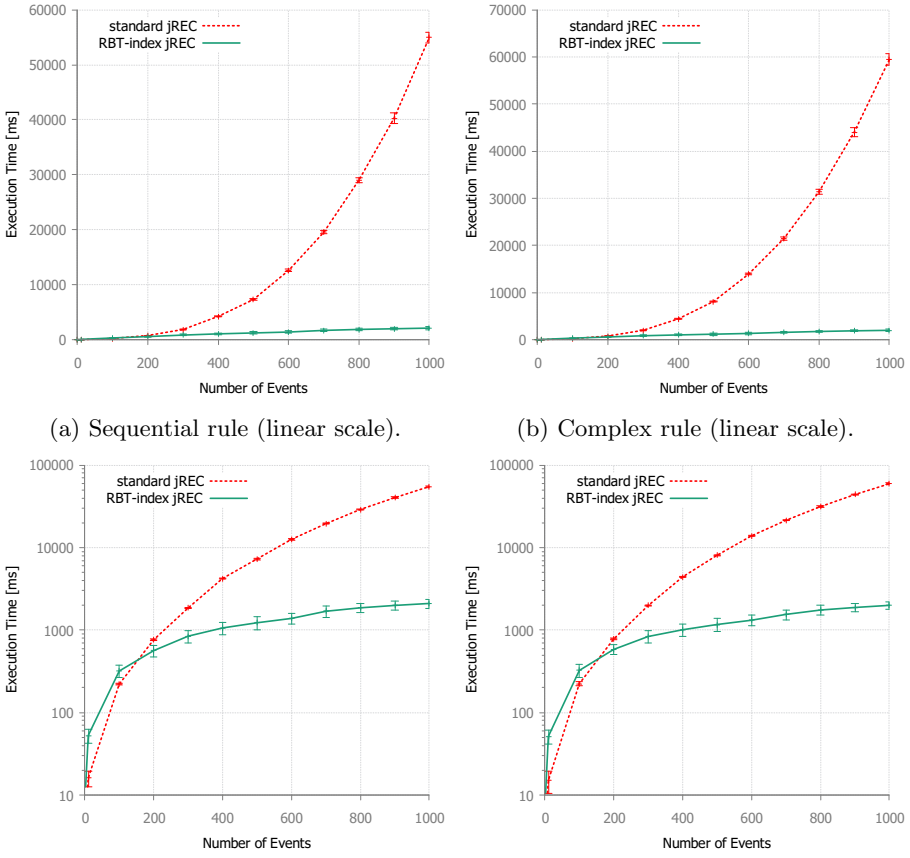
4.2 Results and Discussion

The tests have been executed on an i7-6700K@4.20 GHz CPU with 16 GB@2400 MHz DDR4 RAM, running Ubuntu/Linux 16.04 and Java Runtime Environment 8u121.

It should be noticed that the main results of these tests are the execution time trends, rather than the absolute values themselves (since they vary with different machines).

From the plots in Figure 2a and 2b, it is clear that the two agent minds show a very different behaviour: the execution time of the standard jREC agent mind grows in a polynomial fashion, and is considerably higher than the counterpart's. In fact, from the plots in Figure 3a and 3b it can be observed that the execution time of the RBT-index jREC agent mind follows a logarithmic-like curve.

The polynomial trend exhibited by the standard jREC agent mind can be explained in terms of nested sortings: in fact, every time the engine knowledge base is updated with one or more events, the logic machinery of the engine launches multiple nested sorting clauses (see also Section 3.3).



(a) Sequential rule (linear scale).

(b) Complex rule (linear scale).

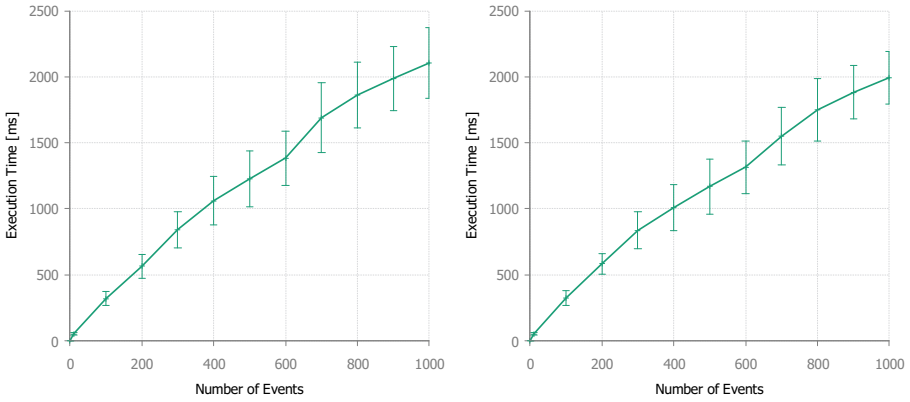
(c) Sequential rule (log scale).

(d) Complex rule (log scale).

Fig. 2: Milliseconds needed by the two jREC agent minds to compute an alert, for the different rules.

On the other hand, the logarithmic-like trend of the RBT-index jREC agent mind highlight a direct correlation with the expected performances of the RBT-based indexing. It also demonstrates that, as the number of event grows, the execution time introduced by the reasoning on such events plays a minor role on the overall execution time. This can be explained by considering that the average number of events falling inside a rule timewindow is constant, since (i) the average event inter-arrival time and (ii) rules time-windows duration are fixed.

As a last remark on the performance gap, the logarithmic plots in Figure 2c and 2d clearly highlight that, after 1000 events, the improvement almost reaches 2 orders of magnitude.



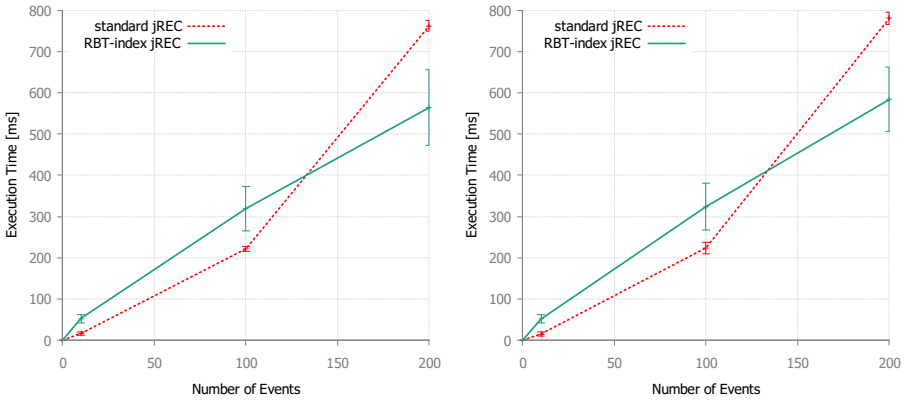
(a) Sequential rule.

(b) Complex rule.

Fig. 3: Detail of Figure 2a and 2b, showing only RBT-index jREC agent mind's trends.

The execution times of the two agent minds in a scenario with a small number of events highlights some peculiar behaviours. As can be clearly seen from the plots (Figure 4a and 4b), when the number of events is smaller than 100, the standard jREC agent mind shows better performances than the other. When the number of events reaches 200, the situation is the opposite, with the RBT-index agent mind being on top. This effect is due to the additional overhead in RBT-index agent mind: to be more precise, with very few events, the event-handling time gain obtained with the exploitation of the RBT-based indexing is not enough to compensate the additional overhead introduced by such data structure. By increasing the number of events, the event-handling time gain becomes increasingly more predominant over the data structure overhead, allowing the RBT-index jREC agent mind to perform better on massive event streams.

With the machine used for the tests, it is shown that the RBT-index jREC agent mind exhibits a clear performance improvement over the standard jREC



(a) Sequential rule.

(b) Complex rule.

Fig. 4: Detail of Figure 2a and 2b, with number of events going from 0 to 200.

agent mind. The execution time trends suggest that even scenarios with more than a thousand events are reasonably manageable by the RBT-index agent mind. Thus, some possible real-case applications for the said agent mind, with the proposed rule patterns, can be:

- Detecting Brittle Diabetes with Continuous Glucose Monitoring devices. They can provide glucose measurements up to one minute [16], so referring to the Rule Pattern 1, it would mean a worst case scenario of 360 events.
- Detecting Pre-Hypertension conditions with digital arm sphygmomanometers. It is enough to have two blood pressure measurements per day [6], so referring to Rule Pattern 2, it would mean a worst case scenario of 14 events.

Even though the standard jREC agent mind performs slightly better with a low number of events, this is not enough to justify its usage only in such scenario.

5 Related work

Multi-Agent Systems (MASs) meet the requirements of the healthcare sector: context awareness, reliability, data abstraction and interoperability, unobtrusiveness [4]. From a requirements engineering perspective, goal-oriented and agent-based design methodologies are useful to tailor pervasive systems to end-users and stakeholders' needs [11]. When appied to PHSs, agent-based modeling has the potential to bring the decision making at the level of self-management of chronic diseases [21]. In the implementation phase, MASs in PHSs pursue the enhancement of home-based self-care by using networks of sensors and remote assistance, to increase the satisfaction of the patient and make an efficient use of resources [17].

Reasoning agents in PHSs allow to transfer part of the knowledge from domain experts to the handheld devices used to perform the self-management of chronic diseases. Beyond PHSs, other applications include energy management [25], to control energy demand and production, home automation [26], to coordinate the available appliances, and ambient assisted living [23, 27], with monitoring purposes. In the context of PHSs, EC and MASs have been successfully applied to the self-management of diabetes [19, 7]. However, such works do not take into account the scalability of the PHS. In fact, a clear advantage of reasoning agents in the Tier-2 of PHSs is the system scalability with increasing number of patients, as showed in [8]. Nevertheless, in such research, the scalability of the agent minds with high streams of events is not considered. Thus, there is the need to find the suitable tools to implement agent minds, which are supposed to run in portable devices, even with high numbers of events and large datasets. This is especially true for EC given its complexity. Indeed, non-logic based pattern recognition has been proved, overall, more efficient than traditional EC when performing predictions. However, it lacks the potential of coding domain experts' knowledge into logic rules and needs to train on large amounts of data. Hence, caching and windowing techniques to make EC efficient and applicable with large scale dataset have been investigated [5, 1]. There are some already available tools that allow logic programming in terms of EC. One of such is DECReasoner [22], a Discrete Event Calculus Reasoner: it implements EC without any caching mechanism and, thus, it is not usable for this research, due to its computation time with the datasets used for the performance tests. A more efficient EC implementation is RTEC, which adds to EC support for handling event streams [1]. However, RTEC techniques such as event windowing and theory pre-compilation do not match the flexibility requirements of the proposed PHS. In addition, it is not compatible with the platform, as the agent-oriented PHS is based on tuProlog and Java [8]. Thus, jREC has been used to implement a prototype for the proposed agent mind, since it implements Cached Event Calculus [5] with tuProlog, a Java-based Prolog engine [15]. Moreover, being Java-based, jREC and tuProlog can run on Android devices, allowing to run the proposed agent mind on handheld devices.

6 Conclusions

In this work, two rule-based minds for monitoring agents running on Tier-2 of a PHS have been presented and tested. Being both integrated into the MAGPIE agent platform, one is based on the plain jREC reasoner and the other is a customization of the standard jREC reasoner augmented with an RBT-based indexing technique. In order to be used in real monitoring scenarios, the agent minds have to be able to process massive event streams, represented by the patient's physiological values. Therefore, in addition to the customization of the jREC reasoning engine, the main contribution of this paper is the performance evaluation of proposed agent minds on the time needed to trigger alerts based on glucose and blood pressure levels, in a diabetes monitoring scenario. Two

real application scenarios for the proposed agent minds are the detection of brittle diabetes, with Continuous Glucose Monitoring, and the detection of Pre-Hypertension conditions, with devices such as digital arm sphygmomanometers.

As future work, since PHSs are intended for the self-management of diabetes with handheld devices, the tests should be performed on mobile phones, to obtain more realistic figures. In this direction, the performance of the RBT-index jREC agent mind can be further enhanced by improving the sophistication of the current indexing solution. Furthermore, in order to validate the usefulness of the rules, the tests should run on real datasets. Lastly, the system can be applied to other use cases, in order to model rules for other diseases.

References

1. Artikis, A., Sergot, M., Paliouras, G.: An event calculus for event recognition. IEEE Transactions on Knowledge and Data Engineering 27(4), 895–908 (2015)
2. Bauer, U.E., Briss, P.A., Goodman, R.A., Bowman, B.A.: Prevention of chronic disease in the 21st century: elimination of the leading preventable causes of premature death and disability in the USA. The Lancet 384(9937), 45–52 (2014)
3. Bayer, R.: Symmetric binary b-trees: Data structure and maintenance algorithms. Acta Informatica 1(4), 290–306 (Dec 1972), https://doi.org/10.1007/BF00289509
4. Bergenti, F., Poggi, A.: Multi-agent systems for e-health: Recent projects and initiatives. In: 10th Workshop on Objects and Agents, WOA'09 (2009)
5. Bragaglia, S., Chesani, F., Mello, P., Montali, M., Torroni, P.: Reactive event calculus for monitoring global computing applications. In: Artikis, A., Craven, R., Kesim Çiçekli, N., Sadighi, B., Stathis, K. (eds.) Logic Programs, Norms and Action: Essays in Honor of Marek J. Sergot on the Occasion of His 60th Birthday, pp. 123–146. Springer Berlin Heidelberg (2012)
6. British Hypertension Society: Home blood pressure monitoring protocol (2017, February 17). Retrieved from http://www.bhsoc.org/files/4414/1088/8031/Protocol.pdf
7. Bromuri, S., Puricel, S., Schumann, R., Krampf, J., Ruiz, J., Schumacher, M.: An expert personal health system to monitor patients affected by gestational diabetes mellitus: A feasibility study. Journal of Ambient Intelligence and Smart Environments 8(2), 219–237 (2016)
8. Brugués, A., Bromuri, S., Barry, M., del Toro, O.J., Mazurkiewicz, M.R., Kardas, P., Pegueroles, J., Schumacher, M.: Processing diabetes mellitus composite events in MAGPIE. Journal of Medical Systems 40(2), 44 (2016)
9. Brugués, A., Bromuri, S., Pegueroles-Valles, J., Schumacher, M.I.: MAGPIE: An agent platform for the development of mobile applications for pervasive healthcare. In: Proceedings of the 3rd International Workshop on Artificial Intelligence and Assistive Medicine (AI-AM/NetMed). pp. 6–10 (2014)
10. Calvaresi, D., Cesarini, D., Marinoni, M., Buonocunto, P., Bandinelli, S., Buttazzo, G.: Non-intrusive patient monitoring for supporting general practitioners in following diseases evolution. In: Ortuño, F., Rojas, I. (eds.) Bioinformatics and Biomedical Engineering: Third International Conference, IWBBIO 2015, Granada, Spain, April 15-17, 2015. Proceedings, Part II, pp. 491–501. Springer International Publishing, Cham (2015)

11. Calvaresi, D., Cesarini, D., Sernani, P., Marinoni, M., Dragoni, A.F., Sturm, A.: Exploring the ambient assisted living domain: a systematic review. Journal of Ambient Intelligence and Humanized Computing pp. 1–19 (2016)
12. Calvaresi, D., Marinoni, M., Sturm, A., Schumacher, M., Buttazzo, G.: The challenge of real-time multi-agent systems for enabling iot and cps. In: Proceedings of IEEE/WIC/ACM International Conference on Web Intelligence (WI'17) (2017)
13. Calvaresi, D., Schumacher, M., Marinoni, M., Hilfiker, R., Dragoni, A.F., Buttazzo, G.: Agent-based systems for telerehabilitation: strengths, limitations and future challenges. In: Proceedings of the 10th Workshop on Agents Applied in Health Care (A2HC 2017) (2017)
14. Chittaro, L., Montanari, A.: Efficient temporal reasoning in the cached event calculus. Computational Intelligence 12(3), 359–382 (1996), http://dx.doi.org/10.1111/j.1467-8640.1996.tb00267.x
15. Denti, E., Omicini, A., Ricci, A.: Multi–paradigm Java–Prolog integration in tuProlog. Science of Computer Programming 57(2), 217 – 250 (2005)
16. Dungan, K.: Monitoring technologies – continuous glucose monitoring, mobile technology, biomarkers of glycemic control. In: De Groot, L.J., Beck-Peccoz, P., Chrousos, G., Dungan, K., Grossman, A., Hershman, J.M., Singer, F. (eds.) Endotext [Internet] (2014)
17. Isern, D., Moreno, A.: A systematic literature review of agents applied in healthcare. Journal of Medical Systems 40(2), 43 (2015)
18. Isern, D., Sánchez, D., Moreno, A.: Agents applied in health care: A review. International Journal of Medical Informatics 79(3), 145–166 (2010)
19. Kafalı, Ö., Bromuri, S., Sindlar, M., van der Weide, T., Aguilar Pelaez, E., Schaechtle, U., Alves, B., Zufferey, D., Rodriguez-Villegas, E., Schumacher, M.I., et al.: Commodity12: A smart e-health environment for diabetes management. Journal of Ambient Intelligence and Smart Environments 5(5), 479–502 (2013)
20. Kowalski, R., Sergot, M.: A logic-based calculus of events. New Generation Computing 4(1), 67–95 (1986)
21. Montagna, S., Omicini, A., Angeli, F.D., Donati, M.: Towards the adoption of agent-based modelling and simulation in mobile health systems for the self-management of chronic diseases. In: Proceedings of the 17th Workshop "From Objects to Agents", Catania, Italy, July 29-30, 2016. pp. 100–105 (2016)
22. Mueller, E.T.: Commonsense Reasoning: An Event Calculus Based Approach. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2 edn. (2015)
23. Nefti, S., Manzoor, U., Manzoor, S.: Cognitive agent based intelligent warning system to monitor patients suffering from dementia using ambient assisted living. In: 2010 International Conference on Information Society. pp. 92–97 (2010)
24. Peine, A., Moors, E.H.: Valuing health technology – habilitating and prosthetic strategies in personal health systems. Technological Forecasting and Social Change 93, 68–81 (2015), science, Technology and the "Grand Challenge" of Ageing
25. Ramchurn, S.D., Vytelingum, P., Rogers, A., Jennings, N.: Agent-based control for decentralised demand side management in the smart grid. In: The 10th International Conference on Autonomous Agents and Multiagent Systems-Volume 1. pp. 5–12 (2011)
26. Ruta, M., Scioscia, F., Loseto, G., Sciascio, E.D.: Semantic-based resource discovery and orchestration in home and building automation: A multi-agent approach. IEEE Transactions on Industrial Informatics 10(1), 730–741 (2014)
27. Sernani, P., Claudi, A., Dragoni, A.: Combining artificial intelligence and netmedicine for ambient assisted living: A distributed BDI-based expert system. International Journal of E-Health and Medical Communications 6(4), 62–76 (2015)

28. Silverman, B.G., Hanrahan, N., Bharathy, G., Gordon, K., Johnson, D.: A systems approach to healthcare: Agent-based modeling, community mental health, and population well-being. Artificial Intelligence in Medicine 63(2), 61–71 (2015)
29. Tartarisco, G., Baldus, G., Corda, D., Raso, R., Arnao, A., Ferro, M., Gaggioli, A., Pioggia, G.: Personal health system architecture for stress monitoring and support to clinical decisions. Computer Communications 35(11), 1296–1305 (2012)
30. Touati, F., Tabish, R.: U-healthcare system: State-of-the-art review and challenges. Journal of Medical Systems 37(3), 9949 (2013)
31. Varshney, U.: Pervasive healthcare and wireless health monitoring. Mob. Netw. Appl. 12(2-3), 113–127 (2007)
32. World Health Organization: Global Report on Diabetes. World Health Organization, Geneve (2016)
33. Zimmet, P., Alberti, K.G., Magliano, D.J., Bennett, P.H.: Diabetes mellitus statistics on prevalence and mortality: facts and fallacies. Nature Reviews Endocrinology 12(10), 616–622 (2016)