Indexing the Event Calculus: towards practical human-readable Personal Health Systems

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[4,5]

[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

[5] BISS Institute, Heerlen, the Netherlands
stefano.bromuri@brightlands.com

Abstract. Personal Health Systems (PHS) are mobile solutions tailored to monitoring patients affected by chronic non communicable diseases. In general, a patient affected by a chronic disease can generate large amounts of events: for example, in Type 1 Diabetic patients generate several glucose events per day, ranging from at least 6 events per day (under normal monitoring) to 288 per day when wearing a continuous glucose monitor (CGM) that samples the blood every 5 minutes for several days. Just by itself, without considering other physiological parameters, it would be impossible for medical doctors to individually and accurately follow every patient, highlighting the need of simple approaches towards querying physiological time series. Achieving this with current technology is not an easy task, as on one hand it cannot be expected that medical doctors have the technical knowledge to query databases and on the other hand these time series include thousands of events, which requires to re-think the way data is indexed.
Anyhow, handling data streams efficiently is not enough. Domain experts' knowledge must be explicitly included into PHSs in a way that it can be easily readed and modified by medical staffs. Logic programming represent the perfect programming paradygm to accomplish this task. In this work, an Event Calculus-based reasoning framework to standardize and express domain-knowledge in the form of monitoring rules is suggested, and applied to three different use cases. However, if online monitoring has to be achieved, the reasoning performance must improve dramatically. For this reason, three promising mechanisms to index the Event Calculus Knowledge Base are proposed. All of them are based on different types of tree indexing structures: k-d trees, interval trees and red-black trees. The

paper then compares and analyzes the performance of the three indexing techniques, by computing the time needed to check different type of rules (and eventually generating alerts), when the number of recorded events (e.g. values of physiological parameters) increases. The results show that customized jREC performs much better when the event average inter-arrival time is little compared to the checked rule time-window. Instead, where the events are more sparse, the use of k-d trees with standard EC is advisable.
Finally, the Multi-Agent paradigm helps to wrap the various components of the system: the reasoning engines represent the agent minds, and the sensors are its body. The said agents have been developed in MAGPIE, a mobile event based Java agent platform.

1 Introduction

The incidence of chronic diseases in the population is recognized as a major challenge for the healthcare sector [3]. For instance, the number of people affected by diabetes has doubled in the last 20 years [49]. 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 [48].

Personal Health Systems (PHSs) aim at supporting the self-management of chronic diseases and reducing the healthcare costs by supporting medical doctors in following the patients' disease evolution [13]. PHSs implement the "*healthcare to anyone, anytime, and anywhere*" paradigm, by increasing both the coverage and the quality of healthcare [47]. In fact, PHSs bring the health technology to domestic environments, by customizing healthcare services to the specific needs, practices, and situations of people and their social contexts [37]. PHSs ensure the continuity of care, focusing on a knowledge-based approach integrating past and current data of each patient together with statistical evidence [45]. A PHS is composed of three tiers [46]: 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.

In addition to the modeling capabilities of agent-based frameworks [42] and despite their still open challenges [16, 15], 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 [29], thus being ideal for PHSs. An agent-based platform called MAGPIE [11] 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. In this work, the agent minds are based on an Event Calculus (EC) engine, and are the part of the agent that check the data collected from the body to perform the monitoring task and trigger alerts for the medical doctors 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 considered 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 three agent minds for the MAGPIE agent platform presented in [11]. Two of them have been implemented directly in tuProlog, while the other one has been developed on top of jREC, a Cached Event Calculus (CEC) reasoner based on Java and tuProlog [8]. Having the goal of avoiding unnecessary computations, moving computational complexity from query to update time, CEC accomplished that by caching the maximum validity intervals for fluents (i.e. a condition or a property that can change over time). Even though all the proposed agent minds are integrated into the MAGPIE agent platform and relate to CEC, the caching mechanisms are not the same: for two of them a tree indexing data structure is directly responsible for caching the EC, while for the other, caching already a part of the jREC machinery. So, for the jREC agent mind, tree indexing is not used for caching, but rather for maintaining and accessing ordered data (i.e. events).

As one of the main contributions of the paper, the performance of the three agent minds are evaluated on the time required to trigger an alert, when the number of events generated by the agent body increases. Other relevant contributions are the three EC indexing techniques themselves, their integration into the MAGPIE agent platform and the medical domain-knowledge modeling based on monitoring rules patterns/templates.

The rest of the paper is organized as follows. Section 3 presents the paper background on EC, CEC, jREC and tree indexing data structures. Section 4 introduces to the problems that arise when massive streams of events have to be handled by reasoning engines, and provides a description on the EC indexing techniques as a solution. Section 5 describes an overview of the entire PHS in which the agent minds runs and shows the encoding of possible monitoring rules that check alerts based on glucose, blood pressure levels and heart beats. Section 6 present the experimental results to evaluate the agent minds. Section 2 describes the work related to the presented research. Section 7 draws the conclusions of the paper and outlines the future work.

2 Related Work

Multi-Agent Systems (MASs) meet the requirements of the healthcare sector: context awareness, reliability, data abstraction and interoperability, unobtrusiveness [7]. From a requirements engineering perspective, goal-oriented and agent-based design methodologies are useful to tailor pervasive systems to end-users and stakeholders' needs [14]. When appied to PHSs, agent-based modeling has the potential to bring the decision making at the level of self-management of chronic diseases [32]. 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 [28].

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 [39], to control energy demand and production, home automation [40], to coordinate the available appliances, and ambient assisted living [35, 41], with monitoring purposes. In the context of PHSs, EC and MASs have been successfully applied to the self-management of diabetes [30, 10], and more generally to patient's monitoring [8, 19, 20]. Still in the medical domain, [38] proposes a symbolic representation of ECG waves and uses it for cardiac arrythmia recognition by means of Prolog rules (the ECG waves dataset in EC form can be found at [1]). 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 [11]. 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 [8, 2]. There are some already available tools that allow logic programming in terms of EC. One of such is DECReasoner [33], 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 [2]. However, RTECs techniques such as event windowing and theory pre-compilation do not match the flexibility requirements (i.e. editing agent's theories at runtime) of the proposed PHS. In addition, it is not compatible with the platform, as the agent-oriented PHS is based on tuProlog and Java [11]. Dealing with the aspect of improving EC reasoning feasibility, [8] introduces a new Cached EC implementation based on tuProlog, wrapped in a tool called jREC. On the same line of making the use

of EC practical even with thousand of events, [9] proposes an EC theory, also developed on tuProlog, which achieves fluents and events caching by exploiting a k-d tree indexing data structure.

3 Background

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

3.1 Event Calculus

In a PHS context, the possibility to extend and modify the Knowledge Base at runtime, without the need of rebuilding any binary file, gives logic-based systems an advantage over procedural/imperative approaches when it comes to extensibility, flexibility and customization capabilities. Especially when considering the latest European GDPR regulations that enforce algorithmic fairness and AI explainability [27], logic-based systems that are able to provide result interpretability and code readibility are favorable over traditional Machine Learning reasoning techniques.

Among a set of possible formalisms, the Event Calculus has been chosen as the reference one for this work, as it combines great expressiveness with feasibility of reasoning, intuitiveness, readability and resource availability (implementation and literature) [33, 8, 2]. More precisely, being a logic formalism for reasoning about actions and their effects in time [31], it is a suitable tool for modeling expert systems representing the evolution in time of an entity by means of the production of events. It shall be also noticed that the EC have been also already proposed and used as a reasoning framework for monitoring purposes as in [11, 19, 20].

From a technical point of view, 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:

$$\begin{aligned} &\text{holdsAt}(F = V, 0) \leftarrow \\ &\qquad \text{initially}(F = V). \end{aligned} \tag{1}$$

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

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

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

$$\begin{aligned} &\text{broken}(F = V, [T_{min}, T_{max}]) \leftarrow \\ &\qquad \text{terminatesAt}(F = V, T), \\ &\qquad T_{min} < T, \\ &\qquad T_{max} > T. \end{aligned} \tag{3}$$

$$\begin{aligned} &\text{broken}(F = V_1, [T_{min}, T_{max}]) \leftarrow \\ &\qquad \text{initiatesAt}(F = V_2, T), V_1 \neq V_2, \\ &\qquad T_{min} < T, \\ &\qquad T_{max} > T. \end{aligned} \tag{4}$$

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.

$$\begin{aligned}&\text{holdsFor}(F = V, [T_{min}, T_{max}]) \leftarrow \\ &\qquad \text{initiatesAt}(F = V, T_{min}), \\ &\qquad \text{terminiatesAt}(F = V, T_{max}), \\ &\qquad \text{not broken}(F = V, [T_{min}, T_{max}]).\end{aligned} \tag{5}$$

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

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

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

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

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.

3.1.1 Cached Event Calculus and jREC

Straightforward implementations of EC [31] 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 [18], 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 implemented in Java and tuProlog that is based on a lightweight version of CEC known as Reactive Event Calculus (REC) [8]. The use of Java has been an important requirement in order to ensure code portability and seamless integration with the MAGPIE agent platform that hosts the agents minds.

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.

3.2 Indexing Data Structures

With the goal of improving the EC agents minds performance, three different indexing data structures are used. Section 4 will take care of detailing how these achieve this goal by indexing events and/or fluents.

3.2.1 Red-black trees

A red-black tree (RBT) is a well known data structure proposed by Rudolf Bayer in 1972 [4]. 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 [4]. This is made possible thanks to node coloring: 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 coloring 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 property of self-balancing. Even though the obtained

balance is not perfect, it is proven to be good enough to provide the declared performances [4].

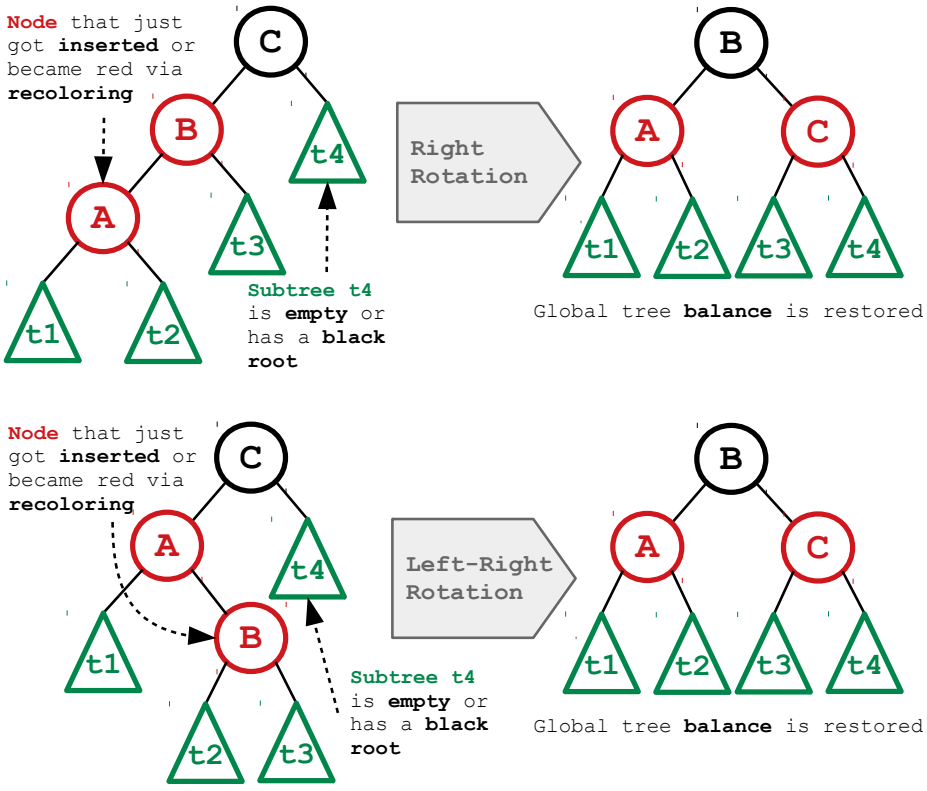


Fig. 1: This picture illustrates how a Red-Black tree keeps its structure balanced by means of right/left rotations and node recoloring. These are triggered when nodes are inserted (or deleted) and the balancing/coloring rules of the data structure are violated

3.2.2 K-d trees

K-d trees [5] are binary trees optimized to deal with k-dimensional points. As reported in [6], given a set of k-dimensional points, we can generate a k-d tree by splitting recursively the hyperplane containing the points at every level of the tree, alternating the coordinate that is split according to the depth of the tree. Fig. 2 shows how splits are performed on a 2-dimensional tree of depth 3, where at each level the value of the splitting coordinate is the median value, deciding if a new point should go to the left or to the right of an existing tree node.

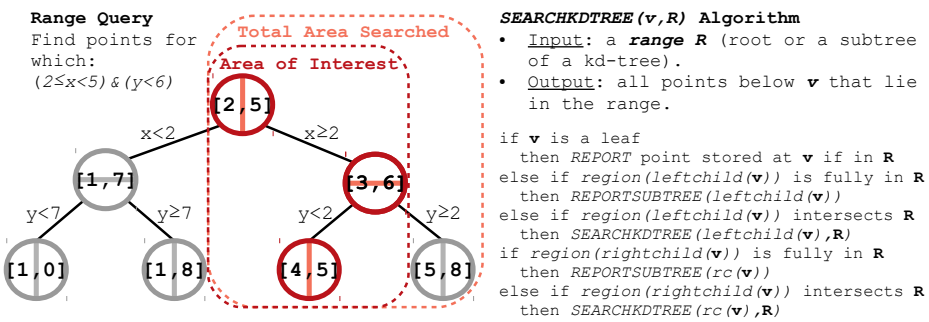


Fig. 2: Range Query on a k-d tree [6]

Fig. 2 shows also the effect of searching a k-d tree via a range query performed on it. The range query algorithm recursively searches for regions contained or intersected by the region specified in the range query. If the region found is contained in the region specified in the query, then the whole region is returned. If the region of the tree intersects the region specified by the query, then the points reported are only those ones included in the region of the query.

The k-d tree data structure has a set of important properties when dealing with searches of multi-dimensional points: (a) a k-d tree for a set P of n points uses $O(n)$ storage and can be constructed in $O(nlog(n))$ time; (b) the operations of adding or deleting a point have a complexity of $O(log(n))$; (c) a rectangular range query on the k-d tree takes $O(\sqrt{n}+k)$ time, where k is the number of reported points residing the rectangular area identified by the query.

These properties are fundamental to create a version of the EC that can scale up to be used in dynamic applications with large narratives.

3.2.3 Interval trees

An interval tree is a data structure that provides efficient means for querying time intervals, and find out what events occurred during a given time interval. In terms of computational complexity, an interval tree composed of n points requires $O(n)$ space for storage; its construction requires $O(nlog(n))$ time; and a query requires $O(log(n)+m)$ time, where n is the total number of intervals and m the number of reported results. Because of these properties, an interval tree is a suitable structure for scaling up EC.

3.3 Medical Domain-Knowledge

One of the contributions of this work is to use the EC language to model different medical aspects that might be of interest in a PHS/self-monitoring scenario.

Diabetes has been chosen as the main use case for the proposed agent minds, as a sustainable and personalized treatment of this disease is considered to be one of the major challenges for the health sector in the next future [48, 49, 17]. More precisely known as Diabetes Mellitus (DM), it is actually a group of metabolic

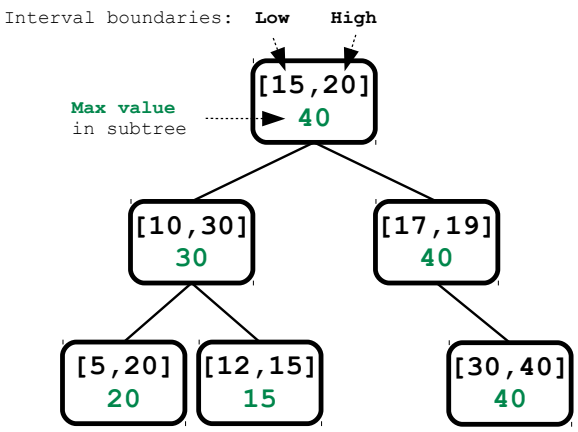


Fig. 3: Depiction of how intervals are stored inside an Interval Tree. Notice that intervals in the same subtree do overlap, while intervals in different subtrees do not. This feature is particularly useful to store and retrieve fluents' MVIs that might be affected by an upcoming event

disorders causing high blood glycemic levels for prolonged periods of time. The disease is usually divided into classes, with the most common two being the following:

- Type 1 Diabetes Mellitus (T1DM), in which a regular external source of insuline is needed, which is normally supplied by manual injections or insulin pumps coupled with Continuous Glucose Monitoring (CGM) devices.
- Type 2 Diabetes Mellitus (T2DM), which instead is associated with elderly people, and usually treated with diet, physical exercise and Self-monitoring of Blood Glucose (SMBG) eventually followed by metforming intake and manual insulin injections.

As the whole field of diabetes is very broad, this work focuses on a particularly hard to control T1DM called Brittle Diabetes, in which the patients experience very big swings in glucose levels, alternating periods of hypoglycemia and hyperglycemia [21].

Another critical task is to monitor physiological parameters that are considered to be risk factors especially for Diabetes chronic patients, such as body weight, cholesterol and blood pressure [24, 17]. Blood pressure control, and more precisely hypertension monitoring, has been chosen as another use case for the proposed agent minds, since it combines high relevance in terms of prevention with low invasiveness of measurements.

The last use case employed to stress the modeling capabilities of EC is cardiac arrythmia. Such as in the case of Diabetes, cardiac arrythmia is a chronic condition that offers a very broad spectrum of diagnoses depending on the shape and frequency of hearth waves. By analyzing the most common type of waves (the P wave and the QRS complex), it is for example possible to distinguish between

the two most straightforward types of arrhythmia: sopraventricular tachycardia and sinusal bradycardia. Sopraventricular Tachycardia is usually defined as a cardiac frequency over 100 beats/minute (or a time less than 100ms between two QRS complexes), while, on the contrary, sinusal bradycardia corresponds to a cardiac frequency less than 60 beats/minute (or a time more than 1s between two QRS complexes) [43].

4 Methods

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

4.1 System Requirements

The main hurdle that hampers the diffusion of logic-based systems in the practical world is efficiency. Even though imperative paradygms usually allow better performance, imperative programs are also usually compiled, thus less flexible and readable than logic theories. For this reason, the design of the proposed optimized EC system have to:

- be Scalable. The computation times for queries must not explode as the Knowledge Base grows.
- provide Human Readable knowledge representation, easing domain-knowledge translation and the presence of humans-in-the-loop.
- be Flexible, allowing for runtime customization without having to stop the execution or either recompiling the code.

4.2 Knowledge-Base Indexing Strategies

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 [2], (ii) theory pre-compilation [2] 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.

The idea used in this work is to separate the Knowledge Base (KB) management from the reasoning itself, as already proposed by [9]. The KB is implemented by an indexing data structure, which will take care of storing and accessing Prolog facts for later and more efficient retrieval. For the reasoning part on the other hand, the EC Prolog engine has to query the indexer every time it wants to read/write some data from/to the KB. The indexing can be performed on many criteria, but since Event Calculus rules heavily depend on temporal constraints expressed as events and fluents, the most reasonable and general choice would be to index events and MVIs on a temporal basis.

4.2.1 Standard EC Indexing

To create a scaled up version of standard EC, tree indexing structures such as k-d trees [9] and interval trees are used to substantially reduce the Prolog computation overhead that arises from handling massive event streams. Even though the introduction of these two different techniques leads to the creation of two distinct EC agent minds, the EC logic machinery is provided with a generic interface that abstracts the communication to those tree structures.

These agents minds are based on a k-d Tree Indexer and an Interval Tree Indexer, both used to store ground terms, but with different internal representations:

- For the k-d Tree Indexer, events and fluent are considered as k-dimensional points. Assuming the events to be instantaneous, they are indexed with respect to one time dimension. Since MVIs are instead intervals, the time dimensions used for indexing are two.
- For the Interval Tree Indexer, events are stored as intervals whose start and end points are the same, while MVIs have different start and end times. In addition, the intervals contain metadata for their identification such as the name of the event, or the name and value of the MVI's fluent.

By the way, one common point among the two indexers is that they keep the event terms and the MVI terms in two separate trees.

A standard EC implementation is enriched with additional predicates that interact with the indexers to insert, delete or range query points. It should be noticed that, since the interface towards the two indexers is the same, the following implementation will refer to a generic tree data structure. The insertion of an event is done with the following predicate,

$$
\begin{aligned}
&\text{insert}(Ev, T) \leftarrow \\
&\qquad \text{index}(Ev, T), \text{cache}(Ev, T). \\
&\text{index}(Ev, T) \leftarrow \\
&\qquad \text{functor}(Ev, Name, Arity), \\
&\qquad Ev = ..[Name|Args], \\
&\qquad Args = [Head|Tail], \\
&\qquad \text{index\_tree}(Ev, Name, Arity, Head, T). \\
&\text{cache}(Ev, T) \leftarrow \\
&\qquad \text{findall}(\text{mvi}(F = V, T_s), \\
&\qquad\qquad \text{terminatesAt}(F = V, T), ListTerm), \\
&\qquad \text{close\_interval}(ListTerm, T), \\
&\qquad \text{findall}(\text{mvi}(F = V, T), \\
&\qquad\qquad \text{initiatesAt}(F = V, T), ListInit), \\
&\qquad \text{open\_interval}(ListInit, T).
\end{aligned}
\tag{9}
$$

The insert/2 predicate consists on two steps. First, the index/2 predicate stores the event in the tree with the index_tree/5 predicate. Second, the cache/2

predicate checks whether the just indexed event is closing or opening the MVI of any fluent F, which is then cached in a separate tree. A MVI can be open to infinity or closed within two time boundaries. The close_interval/2 predicates do the operation of closing open MVIs when needed, such predicates are specified as follows,

$$\begin{aligned}
&\text{close_interval}([], T).\\
&\text{close_interval}([Head|Tail], T) \leftarrow\\
&\qquad Head = \text{mvi}(F = V, _),\\
&\qquad \text{range_query}(F = V, [T_s, +\infty]),\\
&\qquad \text{delete_tree}(F, V, T_s, +\infty),\\
&\qquad \text{index_tree}(F, V, T_s, T).\\
&\text{close_interval}([Head|Tail], T) \leftarrow\\
&\qquad \text{close_interval}(Tail, T).
\end{aligned} \tag{10a}$$

First, the range_query/2 predicate retrieves from the tree all the open MVIs that must be closed due to the happening of the just indexed event. Second, the delete_tree/4 event predicate deletes the MVI from the tree. Finally, the indexed_tree/4 indexes the MVI as closed in the tree. The process for indexing open periods is similar with the difference that no open periods are retracted from the tree. Once the procedure of adding the new event to the rule engine has finished, the next step is to check whether the event is triggering any of the domain dependent rules. In EC notation this means to query the holds_at/2 predicate, which is defined as follows,

$$\begin{aligned}
&\text{holdsAt}(F = V, T) \leftarrow\\
&\qquad \text{not var}(T), \text{number}(T)\\
&\qquad \text{intersect_query}(F = V, [T_{start}, T_{end}], T),\\
&\qquad T > T_{start}, T < T_{end}.
\end{aligned} \tag{11}$$

The intersect_query/3 predicate queries the tree only those MVIs that intersect T. Thus, improving the computation time of the original holdsAt/2 predicate.

Finally, the domain dependent rules that model high level events (or alerts for the matter of this paper) to be notified are expressed in terms of the EC initiatesAt/2 predicate, which is reformulated as follows,

$$\begin{aligned}
&\text{initiatesAt}(F = V, T) \leftarrow\\
&\qquad \text{query_tree}(\text{happensAt}(Ev, T), [WT_S, WT_E]),\\
&\qquad Conditions[T].
\end{aligned} \tag{12}$$

The query_tree/2 predicate queries, to the tree storing the events, only those events that happened within a given time window whose boundaries are WT_S and WT_E. Thus, pruning the number of alternatives that a standard EC implementation would search, which translates to a better performance. In addition,

the query_tree/2 predicate decomposes the different parts of an event in order to perform such queries,

$$\begin{aligned}
&\text{query_tree}(\text{happensAt}(Ev, T), [WT_S, WT_E]) \leftarrow \\
&\qquad \text{var}(T), \text{not var}(Ev), \\
&\qquad \text{functor}(Ev, Name, Arity), \\
&\qquad Ev = ..[Name|Args], \\
&\qquad \text{retrieve_range}(Ev, Name, Arity, Args, T, \\
&\qquad\qquad [WT_S, WT_E]).
\end{aligned} \tag{13}$$

Given the extensive use of tree structures, the proposed agent minds can be wrapped under the name of Tree Event Calculus, as they are both based on tree indexing techniques (i.e. k-d trees and interval trees, thus TEC-KD and TEC-IT).

4.2.2 JREC Indexing

Instead of chaching standard Event Calculus with trees as in the previous section, indexing data structures can be applied to an already more efficient version of EC such as the Reactive Event Calculus, implemented in jREC. In its standard version, 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 fluents' MVIs.

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.

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.

5 Theory

The implemented agent minds run in Tier-2 of the MAGPIE agent-based PHS for self monitoring of physiological values. The entire PHS is depicted in Figure 4. Each patient has its own agent composed by a body and a mind 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.

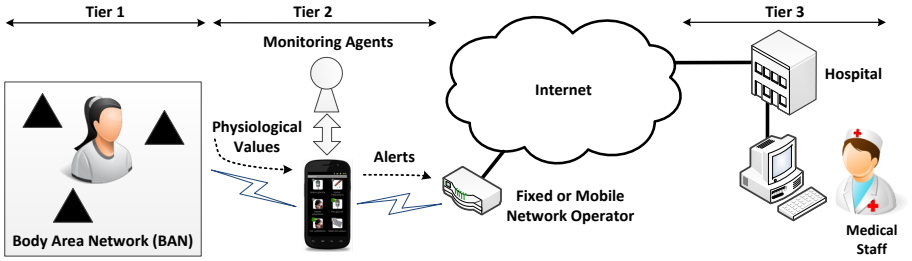


Fig. 4: The agentified PHS. The agent mind runs in Tier-2, to monitor the patient's physiological values.

5.1 MAGPIE Agent Platform

Multi-agent systems have been proven to be very useful Knowledge-Engineering tools, having very versatile modeling capabilities [40]. The MAS paradygm is also particularly suitable when it comes to implement a Personal Health System, thus having to implement features such as interoperability, load balancing, resource allocation, distributed computation and modularity [28]. For this reason, the proposed agent minds have been developed within MAGPIE, an agent platform available for both desktop PCs and mobile (Android) devices. It plays the role of Tier-2 in a PHS by connecting the patient and the medical doctor, with the aim of improving the management of chronic diseases. From the side of the patient it collects physiological values, whereas from a Knowledge Representation side it models the medical knowledge in terms of monitoring rules expressed as domain dependent axioms of EC. All the details about the interaction between the system components, such as the agents' minds, bodies and the PHS's Tier-3 have been given in [11], while a description of the MAGPIE architecture and its integration with Android is available at [12]. In relation to this work, a monitoring rule is defined as a combination of events that trigger an alert to be reported 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.

5.2 Monitoring Rules

In order to detect alert conditions, a sequential and a complex rule patterns are proposed. They check physiological values collected from the patient's BAN, and are based on the literature available for glucose, blood pressure and heartbeat monitoring [26, 34]. Glucose and blood pressure targets for hypoglycemia, hyperglycemia and hypertension have been extracted from diabetes guidelines [25, 22, 23], while thresholds for arrhythmias are taken from [43]. 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 (hypoglycemia) to more than 8.0 mmol/l (hyperglycemia) in a period of six hours. This pattern can be expressed by 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 by a complex rule.

Pattern 3: Tachycardia, defined as nine or more QRS complexes in a period of five seconds. This pattern can be expressed by a complex rule.

Pattern 4: Bradycardia, defined as four or less events of QRS complexes in a period of five seconds. This pattern can be expressed by a complex rule.

5.2.1 JREC Implementation

Pattern 1 is implemented as follows:

$$\begin{aligned}
&\text{initiatesAt}(A = \text{true}, T) :- \\
&\qquad \text{happensAt}(\text{ev}(2, A, W), T), \\
&\qquad \text{happensAt}(\text{ev}(1, A, _), T_1), \\
&\qquad T_s \text{ is } (T - W), \\
&\qquad T > T_1, \\
&\qquad T_1 >= T_s, \\
&\qquad \text{no_alert}(A, T_s).
\end{aligned} \tag{14a}$$

$$\begin{aligned}
&\text{terminatesAt}(A = \text{true}, T) :- \\
&\qquad \text{happensAt}(\text{ev}(1, A, _), T).
\end{aligned} \tag{14b}$$

$$\begin{aligned}\text{happensAt}(\text{ev}(1, \text{`brittle diabetes'}, W), T) :- \\ \text{hours\_to\_epoch}(6, W), \\ \text{happensAt}(\text{glucose}(G), T), \\ G =< 3.8.\end{aligned} \tag{14c}$$

$$\begin{aligned}\text{happensAt}(\text{ev}(2, \text{`brittle diabetes'}, W), T) :- \\ \text{hours\_to\_epoch}(6, W), \\ \text{happensAt}(\text{glucose}(G), T), \\ G >= 8.\end{aligned} \tag{14d}$$

Rules (14a) and (14b) 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 (14c) and (14d) customize the template for the glucose monitoring use case. They instantiate the variables in 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:

$$\begin{aligned}\text{initiatesAt}(A = \text{true}, T) :- \\ \text{happensAt}(\text{alertcheck}(A, W, NMax_1), T), \\ T_s \text{ is } (T - W), \\ \text{count\_events\_tw}(N_1, \text{evc}(1, A), T_s, T), \\ N_1 >= NMax_1, \\ \text{no\_alert}(A, T_s).\end{aligned} \tag{15a}$$

$$\begin{aligned}\text{terminatesAt}(A = \text{true}, T) :- \\ \text{happensAt}(\text{alertcheck}(A, W, \_), T), \\ \text{holdsAt}(A = \text{true}, T).\end{aligned} \tag{15b}$$

$$\begin{aligned}\text{happensAt}(\text{evc}(1, \text{`pre-hypertension'}), T) :- \\ \text{happensAt}(\text{blood\_pressure}(S, D), T), \\ S >= 130, \\ D >= 80.\end{aligned} \tag{15c}$$

$$\begin{aligned}\text{happensAt}(\text{alertcheck}(\text{`pre-hypertension'}, W, 2), T) :- \\ \text{weeks\_to\_epoch}(1, W), \\ \text{happensAt}(\text{evc}(1, \text{`pre-hypertension'}), T).\end{aligned} \tag{15d}$$

Rules (15a) and (15b) 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 (15c) and (15d) 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.

Patterns 3 and 4's have been respectively coded as follows:

$$\begin{aligned}&\text{initiatesAt}(\text{tachycardia} = \text{true}, T) :- \\ &\quad \text{seconds\_to\_epoch}(5, W), \\ &\quad T_0 \text{ is } (T - W), \\ &\quad \text{count\_events\_tw}(N, \text{qrs}(\text{basic}), T_0, T), \\ &\quad N >= 8, \\ &\quad \text{no\_alert}(\text{tachycardia}, T_0).\end{aligned} \tag{16a}$$

$$\begin{aligned}&\text{terminatesAt}(\text{tachycardia} = \text{true}, T) :- \\ &\quad \text{holds\_at}(\text{tachycardia} = \text{true}, T).\end{aligned} \tag{16b}$$

$$\begin{aligned}&\text{initiatesAt}(\text{bradycardia} = \text{true}, T) :- \\ &\quad \text{seconds\_to\_epoch}(5, W), \\ &\quad T_0 \text{ is } (T - W), \\ &\quad \text{count\_events\_tw}(N, \text{qrs}(\text{basic}), T_0, T), \\ &\quad N <= 3, \\ &\quad \text{no\_alert}(\text{bradycardia}, T_0).\end{aligned} \tag{17a}$$

$$\begin{aligned}&\text{terminatesAt}(\text{bradycardia} = \text{true}, T) :- \\ &\quad \text{holds\_at}(\text{bradycardia} = \text{true}, T).\end{aligned} \tag{17b}$$

Unlike the previous jREC implementations of Patterns 1 and 2, in this case rule templates have not been used (thus leading to a less general but more straightforward coding). The "artificial" events (evc) that wrap real ones in rule (15) are not needed anymore, since no condition has to be put on the argument(s) of the qrs/1 event. Apart from that, the mechanism for counting event occurrences (count_events_tw/4) and alert flooding avoidance (no_alert/2) are the same as in (15).

5.2.2 TEC Implementation

Pattern 1 is expressed in the following way:

$$
\begin{aligned}
&\text{initiatesAt}(\text{alert(one)} = \text{`brittle diabetes'}), T) : - \\
&\quad \text{hours\_ago}(6, T_s, T), \\
&\quad \text{query\_tree}(\text{happensAt}(Ev_1, T_1), [T_s, T]), \\
&\quad \text{query\_tree}(\text{happensAt}(Ev_2, T_2), [T_s, T]), \\
&\quad Ev_1 = \text{glucose}(V_1), Ev_2 = \text{glucose}(V_2), \\
&\quad V_1 =< 3.8, V_2 >= 8.0, \\
&\quad T_2 > T_1, \\
&\quad \text{not query\_tree}(\text{happensAt}(\text{alert(one,} \\
&\qquad \text{`brittle diabetes'}), T_{alert}), [T_s, T]).
\end{aligned} \tag{18}
$$

Rule (18) follows the same structure as rule (12). In particular, queries the interval tree for two glucose events, within a time window of six hours, that satisfy the threshold conditions of V_1 and V_2. In addition, it specifies three different temporal conditions, which are: (i) the hours_ago/3 predicate that defines the lower boundary of the time window, (ii) the $T_2 > T_1$ inequality, that ensures the correct temporal ordering between the glucose measurement events and (iii) the last Interval Tree query, which ensures that in the rule's time window there is not the same alert already, to avoid the generation of multiple alerts associated with the same events.

Pattern 2 is implemented as follows:

$$
\begin{aligned}
&\text{initiatesAt}(\text{alert(two)} = \text{`pre-hypertension'}, T) : - \\
&\quad \text{weeks\_ago}(1, T_s, T), \\
&\quad \text{not query\_tree}(\text{happensAt}(\text{alert(two,} \\
&\qquad \text{`pre-hypertension'}), T_{alert}), [T_s, T]), \\
&\quad \text{more\_or\_equals\_to}(2, \\
&\qquad (\text{query\_tree}(\text{happensAt}(Ev_1, T_1), [T_s, T]), \\
&\qquad Ev_1 = \text{blood\_pressure}(Sys_1, Dias_1), \\
&\qquad Sys_1 >= 130, Dias_1 >= 80, \\
&\qquad \text{within\_weeks}(1, T_1, T))).
\end{aligned} \tag{19}
$$

Rule (19) also follows the structure of rule (12). In particular, queries the interval tree for two blood pressure events, within a time window of one week, that satisfy the threshold conditions of Sys_1, $Dias_1$, Sys_2 and $Dias_2$. The predicate more_or_equals_to/2 checks if at least two over-threshold blood pressure events happened inside the time window. The use of this predicate is necessary to handle different event temporal orderings without having to duplicate the rule body for every permutation. In addition, the rule specifies two different temporal conditions, which are: (i) the weeks_ago/3 predicate that defines the lower boundary of the time window and (ii) the first Interval Tree query, which ensures that in the rule's time window there are no other alerts (i.e. avoiding the generation of multiple alerts associated with the same events).

Patterns 3 and 4's have been respectively coded as follows:

$$\begin{aligned}
&\text{initiatesAt}(\text{alert}(\text{three}) = \text{'tachycardia'}, T) :- \\
&\quad \text{seconds_ago}(5, T_s, T), \\
&\quad \text{not query_tree}(\text{happensAt}(\text{alert}(\text{three}, \\
&\qquad \text{'tachycardia'}), T_{alert}), [T_s, T]), \\
&\quad \text{more_or_equals_to}(9, \\
&\qquad (\text{query_tree}(\text{happensAt}(Ev_1, T_1), [T_s, T]), \\
&\qquad Ev_1 = \text{qrs}(\text{basic}), \\
&\qquad \text{within_seconds}(5, T_1, T))).
\end{aligned} \tag{20a}$$

$$\begin{aligned}
&\text{initiatesAt}(\text{alert}(\text{four}) = \text{'bradycardia'}, T) :- \\
&\quad \text{seconds_ago}(5, T_s, T), \\
&\quad \text{not query_tree}(\text{happensAt}(\text{alert}(\text{four}, \\
&\qquad \text{'bradycardia'}), T_{alert}), [T_s, T]), \\
&\quad \text{less_or_equals_to}(4, \\
&\qquad (\text{query_tree}(\text{happensAt}(Ev_1, T_1), [T_s, T]), \\
&\qquad Ev_1 = \text{qrs}(\text{basic}), \\
&\qquad \text{within_seconds}(5, T_1, T))).
\end{aligned} \tag{20b}$$

Indeed, these implementations follow the structure of rule (19). It should be noticed anyway that (20b) makes use of the less_or_equals_to predicate, in contraposition to the other rules implemented in TEC. This predicate allows the rule to check if any five seconds interval contains less than five QRS complexes events, and if so, to generate a bradycardia alert. The regular more_or_equals_to predicate instead appears in (20a), since a tachycardia alert is generated when more than eight QRS complexes happen inside the five seconds time window.

6 Results

The performances of the three agent minds have been evaluated using the sequential and complex rule patterns described above. In order to have a lower bound for execution times, tests with an hardwired Java implementation (for simplicity it will be referred to as NO-EC) of the monitoring rules have been included. 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. As a first step towards validation, Rule Pattern 1 (i.e. the Brittle Diabetes rule) shown section 5.2 has been also tested on real CGM data, by means of the dataset in [44].

6.1 Performance and Scalability Tests

6.1.1 Setup

The tests have been performed on a standard desktop computer running Java 8 on top of Ubuntu 16.04, with 16 GBs of RAM and a i7-6700k CPU. Interested readers can find code and executable files following the link in [36].

The following are the agents minds that have been tested:

- JREC with Red-Black Tree indexing (jREC-RBT);
- Standard Event Calculus cached with k-d trees (TEC-KD);
- Standard Event Calculus cached with interval trees (TEC-IT).

In order to specifically stress them, two different event dataset conditions have been set up:

- The events in the dataset are concentrated in a short period of time, so that all of them happen inside the rule's time window. This will be called the "dense" events condition.
- The events in the dataset are spread across the time axis, so that the rule's timewindow contains only a fraction of them. This will be called the "sparse" events condition.

In this way, 4 test scenarios have been defined to study all the combinations of Rule-Type/Event-Condition:

- Complex Rule/Dense Events
- Complex Rule/Sparse Events
- Sequential Rule/Dense Events
- Sequential Rule/Sparse Events

Then, to appreciate how the performances of the agent minds evolve when the number of events increases, a series of random datasets 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.

For the Sequential Rule/Dense Events scenario, the tests had to be stopped at 500 events, due to too high computation times from the TEC-IT and TEC-KD agent minds. Instead, for all the other scenarios, tests could have been only run up to 3000 events due to the jREC-RBT agent mind consuming too much memory.

For the purpose of these tests, the use of synthetic datasets did not represent a threat to the experiment validity. It instead turned out to be a useful feature, since it allowed to stress the agent minds on critical tasks. It should be also clear that such datasets have been created to be as realistic as possible, with particular attention to event inter-arrival times and physiological values.

6.1.2 Plots and Description

Absolute values of the execution times needed by an agent mind to trigger an alert is the most direct feature to evaluate the reasoning feasibility in a specific scenario. However, since the agent minds behave very differently depending on the Rule Type/Events Condition combination, the most efficient agent mind to adopt changes from case to case. As execution time absolute values also directly depend on the computer hardware, execution time trends represent a more stable feature across different machines. Such trends can be in fact used as a good insight to evaluate their scalability when the number of events to be handled increases significantly.

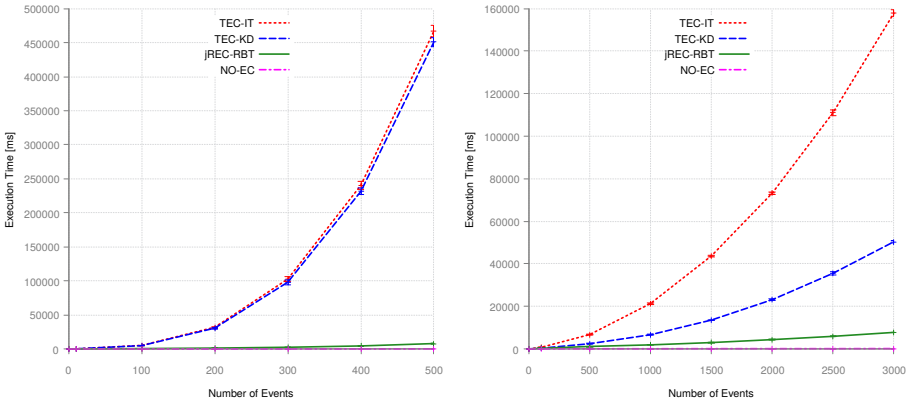
In the Sequential Rule/Dense Events scenario (fig. 5a), the execution time trends for the TEC-IT and the TEC-KD agent minds are shown to be growing very fast, in a (at least) polynomial fashion. The jREC-RBT agent mind is as well showing a polynomial-like trend (fig. 7a), but with a lower growing rate. It is also shown to be faster than the other two agent minds by almost two orders of magnitude (fig. 6a). For the Complex Rule/Dense Events scenario (fig. 5b), the situation is similar, even though less extreme. TEC-IT and TEC-KD agent minds are shown to have polynomial-like trends, with TEC-IT's trend growing significantly faster than TEC-KD's. JREC-RBT is also shown to have a slowly increasing polynomial-like trend (fig. 7b), and to be around one order of magnitude faster than the other two agents minds (fig. 6b).

Plots in figures 5c and 5d highlight a more balanced situation, with TEC-KD being the fastest agent mind among both Sequential and Complex Rule type. In the Sequential Rule/Sparse Events scenario (fig. 5c) TEC-IT turns out to be the slowest agent mind, with jREC-RBT showing just slightly faster execution times. In the Complex Rule/Sparse Events (fig. 5d), the situation for the said two agent minds is the opposite, with jREC-RBT being the slowest one. Also the execution time trends are quite similar. In both figures 5c and 5d TEC-IT and TEC-KD exhibit a linear-like trend, with TEC-KD's trend slope being consistently lower than TEC-IT's. On the other hand, the jREC-RBT trend is less encouraging, as it shows a slowly growing polynomial behavior.

6.1.3 Discussion

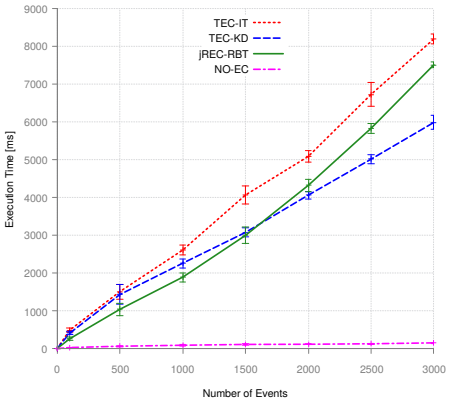
The reason for the agent minds in the Dense Event condition being usually slower than in the Sparse Event condition lies in how many events occurr inside the rule's time-window. The more this number increases, the longer it will take for the agent mind to check the rule (the events outside the rule's time-window are not checked). In other words, a crucial parameter that affects agent minds' performance can be defined as the ratio bewteen the average event inter-arrival time and the duration of a rule time-window..

From the perspective of Rule Types, the explanation for why the Sequential Rule is more difficult to check than the Complex, can be found on event ordering. For the Sequential Rule in fact, the agent minds not only have to check if some events did happen, but also whether or not they happened in a particular order.

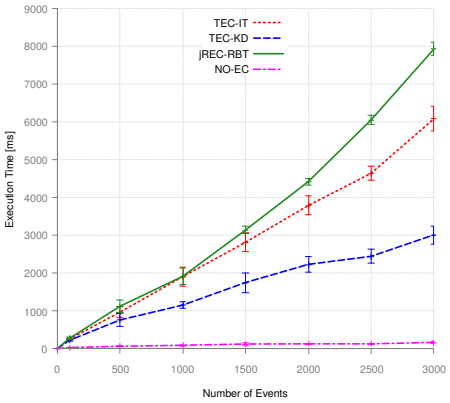


(a) Sequential Rule/Dense Events.

(b) Complex Rule/Dense Events.



(c) Sequential Rule/Sparse Events.

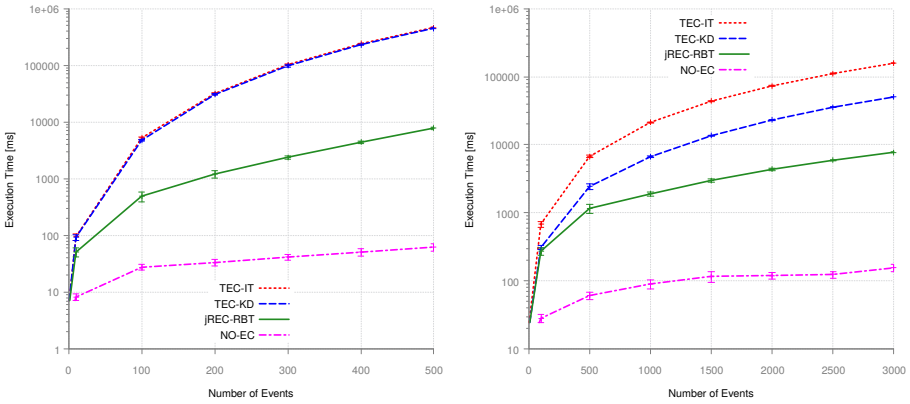


(d) Complex Rule/Sparse Events.

Fig. 5: Milliseconds needed by the three EC agent minds to compute an alert, for all the Rule-Type/Event Condition combinations.

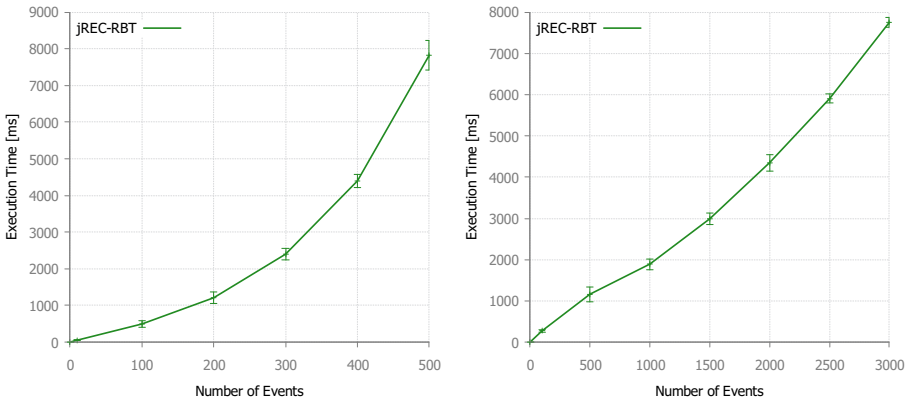
Thus, by combining these consideration, it is clear why the Sequential Rule/Dense Events represents the worst case scenario for all the proposed agent minds, while the Complex Rule/Sparse Events is the easiest one. Moreover, it can be also asserted that the jREC-RBT agent mind is the most suitable one for the Dense Events condition, while the TEC-KD is the one to choose for the Sparse Events condition.

Unsurprisingly, plots in figure 5 and figure 6 show that the NO-EC implementation outperforms all the logic engines by two orders of magnitude and shows a linear-like trend. This is not only due to the more efficient nature of hardwired solutions, as it also depends on different hypothesis that are made on events timestamps. The proposed logic-based approaches do not apply any simplifying assumption, and can correctly reason on not chronologically ordered data; on the



(a) Sequential Rule/Dense Events (log). (b) Complex Rule/Dense Events (log).

Fig. 6: Logarithmic scale plots of fig. 5a and fig. 5b



(a) Sequential Rule/Dense Events. (b) Complex Rule/Dense Events.

Fig. 7: Detail of fig. 5a and fig. 5b showing only the jREC-RBT agent mind.

other hand, to put the NO-EC engine as the lowest possible baseline, it has been implemented in such a way that the events have to be fed in chronological order to obtain correct results. In most monitoring scenarios it should be technically possible to keep this latter assumption and modify the logic-based reasoners accordingly, thus trading reasoning generality with performance. Anyway, for this work, in order to keep the PHS as flexible and as expressive as possible, the use of hardwired or Complex Event Processing solutions should be avoided, and the assumption on chronologically ordered data not applied.

6.2 Tests on CGM Dataset

Now that the overall engine performance and scalability properties have been inquired, a first step towards a system validation and a real world deployment consists on observing how the proposed rule patterns behave on real data. Hence, for this purpose, the Brittle Diabetes rule in section 5.2 has been tested on a publicly available Continuous Glucose Monitoring dataset [44], which contains hundreds of thousand of blood glucose recordings[1] from hundreds of patients. In the format of csv files, this dataset is essentially a set blood glucose logs, providing the measurements recorded by digital CGM devices for several patients, and in multiple time slots. As expected, each patient's blood glucose has been measured and recorded every 5 minutes, for periods that range from a minimum of about 2 to a maximum of 20 hours. After getting rid of few errors present in the data, the longest recording batches for each patient have been extracted, and the Brittle Diabetes rule has been tested on top of them.

6.2.1 Plots and Discussion

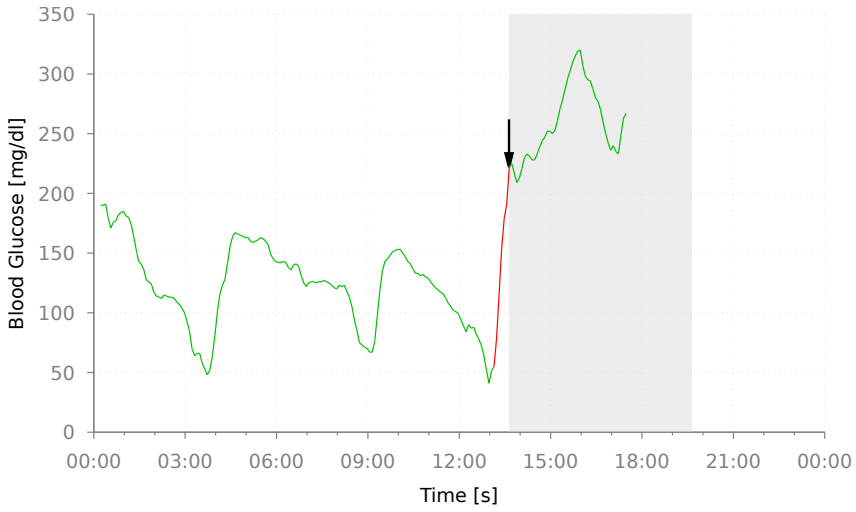


Fig. 8: Single Brittle Diabetes Alert

Among all the plots extracted from the dataset, just two of the most relevant ones have been reported.

[1] In this dataset, Blood Glucose measurements are found in a different unit of measure. Thus, for this test, target values in the Brittle Diabetes rule (see section 5.2) have been converted from $mmol/L$ to mg/dL.

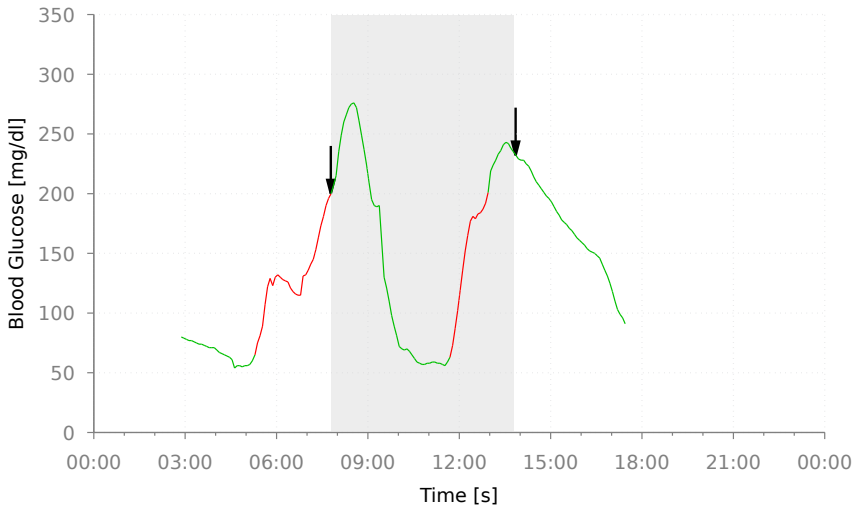


Fig. 9: Double Brittle Diabetes Alert

The red sections of the plots lines highlight the sequences of blood glucose events that triggered the Brittle Diabetes rule, and the arrows indicate the moment in which the alerts are generated. The grey regions represent instead the periods of time in which the alert generation is disabled, by means of the no-alert condition. This condition is necessary in order to avoid alert flooding, or in other words, to prevent that each new blood glucose event generates an additional alert (see implementation in sections 5.2.1 and 5.2.2).

Plots 8 shows a situation in which the rule is triggered only once. As expected, an alert is only generated when the blood glucose swings below and above the thresholds, inside the time window defined in section 5.2. Plot 9 shows instead how the agent minds behave when a double swing occurs. The first swing generates an alert normally, but, what should be noticed is that the second one is triggered slightly after the above-threshold blood glucose event happens. This latter event falls inside the no-alert window, so this swing's alert is triggered by the first blood glucose event that happens right after the no-alert window termination.

To sum up, these tests have proven that the Brittle Diabetes rule running on the proposed agent minds is suitable for analyzing real data coming from CGM devices. In fact, not only they have been useful to assess that the Brittle Diabetes rule behave well in a realistic scenario in terms on alert generation and alert flooding avoidance, but they've been also helpful to ensure that the number of events is appropriate to have a computationally feasible rule evaluation.

7 Conclusions

In this work, three rule-based minds for monitoring agents running on Tier-2 of a PHS have been presented and tested. Being all integrated into the MAGPIE agent platform, two of them are based on an EC implementation which exploits tree indexing to achieve fluents and events caching. The remaining one is instead a customization of the standard jREC reasoner augmented with an red-black tree 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 proposed EC customizations, 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. The real application scenarios for the proposed agent minds are the detection of brittle diabetes, with Continuous Glucose Monitoring, the detection of Pre-Hypertension conditions, with devices such as digital arm sphygmomanometers, and the detection of cardiac arrhythmia by means of smart ECG holters. The tests have shown that the indexing techniques do improve EC efficiency while preserving the logic programming phylosophy, and are necessary for practical applications. Among the tested techniques, the one that showed the overall better performance was the jREC with the Red-Black Tree indexing. Tests on the real glucose monitoring dataset showed that the rules implementation behaved as expected.

As future work, additional tests will be performed. A part of these will be devoted to evaluate how the ratio between the average event inter-arrival time and the rules' time-window duration affects the reasoning performance. The performance of the RBT-index jREC agent mind will be eventually enhanced by improving the current indexing solution. An improvement might be the creation of an hybrid engine composed of two parts: (i) a machine learning model (e.g. Deep Learning, regression, etc.) that processes raw data/events coming from sensors and synthesizes an higher-level data representation, (ii) and a logic-based (EC) reasoner on which domain-expert's knowledge can be expressed. Since PHSs are intended for the self-management of diabetes with handheld devices, another part of the tests should then focus on mobile phones, to obtain more realistic figures. Lastly, the system can be applied to other use cases, in order to model rules for other diseases.

Acknowledgments

This work is inspired by an original idea of S. Bromuri and K. Stathis, and partially supported by a STSM Grant from COST Action IC1303.

References

1. Alexander Artikis: Datasets for symbolic event recognition. Retrieved from `http://users.iit.demokritos.gr/~a.artikis/ER-datasets.html`

2. Artikis, A., Sergot, M., Paliouras, G.: An event calculus for event recognition. IEEE Transactions on Knowledge and Data Engineering 27(4), 895–908 (2015)
3. 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)
4. 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
5. Bentley, J.L.: Multidimensional binary search trees used for associative searching. Commun. ACM 18(9), 509–517 (1975)
6. de Berg, M., van Kreveld, M., Overmars, M., Schwarzkopf, O.: Computational Geometry: Algorithms and Applications. Springer-Verlag, third edn. (2008)
7. Bergenti, F., Poggi, A.: Multi-agent systems for e-health: Recent projects and initiatives. In: 10th Workshop on Objects and Agents, WOA'09 (2009)
8. 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)
9. Bromuri, S., Brugues de la Torre, A., Duboisson, F., Schumacher, M.: Indexing the Event Calculus with Kd-trees to Monitor Diabetes. ArXiv e-prints (Oct 2017)
10. 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)
11. 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)
12. 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)
13. 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)
14. 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)
15. 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)
16. 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)
17. Centers for Disease Control and Prevention: Diabetes risk factors (2018, 02). Retrieved from https://www.cdc.gov/diabetes/pdfs/data/statistics/national-diabetes-statistics-report.pdf

18. 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
19. Chittaro, L., Montanari, A., Dojat, M., Gasparini, C.: The event calculus at work: a case study in the medical domain. In: Second International Conference on Intelligent Systems Engineering, 1994. pp. 195–200 (Sep 1994)
20. Chittaro, L., Del Rosso, M., Dojat, M.: Modeling medical reasoning with the event calculus: an application to the management of mechanical ventilation. In: Barahona, P., Stefanelli, M., Wyatt, J. (eds.) Artificial Intelligence in Medicine. pp. 79–90. Springer Berlin Heidelberg, Berlin, Heidelberg (1995)
21. Diabetes.co.uk, the global diabetes community: Brittle diabetes (labile diabetes). Retrieved from https://www.diabetes.co.uk/brittle-diabetes.html
22. Diabetes.co.uk, the global diabetes community: Diabetes and hyperglycemia. Retrieved from https://www.diabetes.co.uk/Diabetes-and-Hyperglycaemia.html
23. Diabetes.co.uk, the global diabetes community: Diabetes and hypoglycemia. Retrieved from https://www.diabetes.co.uk/Diabetes-and-Hypoglycaemia.html
24. Diabetes.co.uk, the global diabetes community: National diabetes statistics report. Retrieved from https://www.diabetes.co.uk/Diabetes-Risk-factors.html
25. Diabetes.co.uk, the global diabetes community: Type 1 diabetes in adults: diagnosis and management. Retrieved from https://www.diabetes.co.uk/diabetes_care/blood-sugar-level-ranges.html
26. 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)
27. Goodman, B., Flaxman, S.: European Union regulations on algorithmic decision-making and a "right to explanation". ArXiv e-prints (Jun 2016)
28. Isern, D., Moreno, A.: A systematic literature review of agents applied in healthcare. Journal of Medical Systems 40(2), 43 (2015)
29. Isern, D., Sánchez, D., Moreno, A.: Agents applied in health care: A review. International Journal of Medical Informatics 79(3), 145–166 (2010)
30. 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)
31. Kowalski, R., Sergot, M.: A logic-based calculus of events. New Generation Computing 4(1), 67–95 (1986)
32. 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)
33. Mueller, E.T.: Commonsense Reasoning: An Event Calculus Based Approach. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2 edn. (2015)
34. National Institute for Health and Care Excellence: Hypertension in adults: diagnosis and management (2016, November). Retrieved from https://www.nice.org.uk/guidance/cg127/chapter/1-Guidance#measuring-blood-pressure

35. 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)
36. Nicola Falcionelli: Github repository on indexed event calculus (2018, May). Retrieved from https://github.com/N1k06/event-calculus
37. 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
38. Portet, F.: Algorithms piloting for cardiac arrhythmias recognition. Theses, Université Rennes 1 (Dec 2005), https://tel.archives-ouvertes.fr/tel-00011942
39. 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)
40. 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)
41. 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)
42. 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)
43. Spodick, D.H.: Normal sinus heart rate: appropriate rate thresholds for sinus tachycardia and bradycardia. South. Med. J. 89(7), 666–667 (Jul 1996)
44. T1D Exchange: A randomized trial comparing continuous glucose monitoring with and without routine blood glucose monitoring in adults with type 1 diabetes (2016, September). Retrieved from https://t1dexchange.org/pages/resources/our-data/studies-with-data/
45. 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)
46. Touati, F., Tabish, R.: U-healthcare system: State-of-the-art review and challenges. Journal of Medical Systems 37(3), 9949 (2013)
47. Varshney, U.: Pervasive healthcare and wireless health monitoring. Mob. Netw. Appl. 12(2-3), 113–127 (2007)
48. World Health Organization: Global Report on Diabetes. World Health Organization, Geneve (2016)
49. 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)