Computational Intelligence, Volume , Number 000,

AGENT-ORIENTED ACTIVITY RECOGNITION IN THE EVENT CALCULUS: AN APPLICATION FOR DIABETIC PATIENTS

ÖZGÜR KAFALI
Department of Computer Science, North Carolina State University, USA

ALFONSO E. ROMERO
Department of Computer Science, Royal Holloway, University of London, UK

KOSTAS STATHIS
Department of Computer Science, Royal Holloway, University of London, UK

We present a knowledge representation framework based on the Event Calculus, that allows an agent to recognise complex activities from low-level observations received by multiple sensors, reason about the lifecycle of such activities, and take action to avoid or support their successful completion. Activities are understood as multi-value fluents that change according to events that occur in the environment. The parameters of an activity fluent consist of a unique label, a set of participants involved in the carrying out of the activity, and a unique goal associated with the activity revealing the activity's desired outcome. Our contribution is the identification of an activity lifecycle describing how activities can be started, interrupted, suspended, resumed, or completed over time, as well as how these can be represented. The framework also specifies activity goals, their associated lifecycle, and their relation with the activity lifecycle. We provide the complete implementation of the framework, which includes an activity generator that automatically creates synthetic sensor data in the form of event streams that represent the everyday lifestyle of a type 1 diabetic patient. We also test the framework by generating very large activity streams that we use to evaluate experimentally the performance of the recognition capability and study its relative merits.

Key words: Activity recognition; Diabetes; Event Calculus; Activity generator; Performance evaluation.

1. INTRODUCTION

We study the problem of how to develop an activity recognition capability as part of a healthcare application with the aim of assisting a patient in the monitoring and management of his diabetes. This problem is important because the possibility of delegating parts of the monitoring and management of a diabetic's activity to a software application has the advantage of simplifying the patient's lifestyle. Amongst other things, a patient would not have to worry about where to systematically record regular measurements of his blood glucose, or how to distinguish trends that may determine his well-being and, in the ultimate analysis, his health. This is, however, a complex task because the application must be in position to recognise the patient's activities using sensor technology, relate these activities to medical guidelines that must be reasoned upon and interpreted in conjunction to medical expertise, as well as make suggestions that do not overwhelm the patient with notifications or requests for input information.

We argue that such a challenging application can be naturally developed as a multi-agent system for the following reasons. The problem of monitoring requires a continuous and dedicated software process that observes the condition of the patient. First, this process must also encapsulate its own state, to store information such as glucose measurements or patient profile information. In addition, the process must be both reactive, in order for example to alert the patient about significant events that are relevant to his condition, but also

Address correspondence to Özgür Kafalı, Department of Computer Science, North Carolina State University, Raleigh, NC 27695-8206, USA, `rkafali@ncsu.edu`.

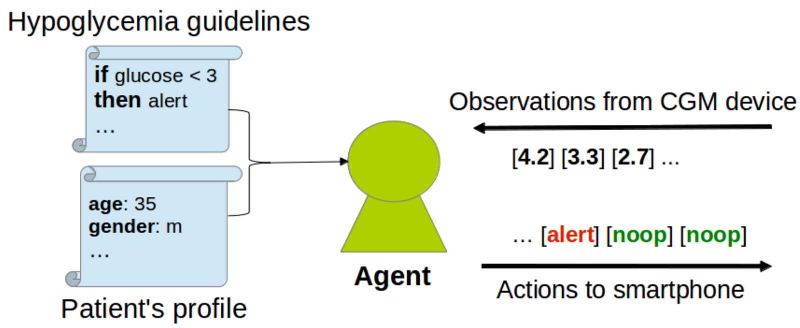


FIGURE 1. Continuous Glucose Monitoring (CGM) Agent in COMMODITY$_{12}$.

proactive, to evaluate the significance of certain events, reason about their effects and choose appropriate action that will be to the benefit of the patient. Furthermore, the process must be also in a position to access and influence the environment via state-of-the-art sensor/actuation technologies, for instance, to measure glucose values or administer insulin respectively. Most importantly, the process should be able to interact and communicate with other similar processes representing the interests of doctors, hospitals, or family members of patients, to inform and alert of critical situations as they arise, and by using specific protocols, sometimes formal and strict, while other times informal and flexible.

From our involvement in the FP7 COMMODITY$_{12}$ project, we have been particularly preoccupied with developing a monitoring agent that is a specialised version of the KGP model (Kakas et al., 2008; Forth et al., 2006). Such an agent diagnoses (Kafalı et al., 2013), ontologically reasons about (Kafalı et al., 2013) and together with specialised agents predict (Kafalı et al., 2014) medical emergencies such as *hypoglycaemia*. According to the International Classification of Diseases (ICD), hypoglycaemia is defined as the patient's glucose level being below a certain threshold value. When it arises, it can produce a variety of symptoms and effects but the principal problems is an inadequate supply of glucose to the brain, resulting in impairment of function and, eventually, to permanent brain damage or death. According to the severity level of hypoglycaemia, a series of actions may need to be taken immediately, including informing the doctor of the patient as soon as possible, to require advice, or to start an emergency protocol.

Patients with diabetes develop an increased risk of cardiovascular disease with both microvascular complications and macrovascular disease. Besides, the average individual with type 1 diabetes experiences about two episodes of symptomatic hypoglycaemia per week, which is a figure that has not been substantially reduced in the last years (see McCrimmon and Sherwin (2010) and references therein). Amongst all the hypoglycaemia episodes, the severe ones (those in which the patient requires help for recovery) have a relatively annual high prevalence (between 30% and 40% of all type 1 diabetic patients suffer at least one severe episode per year). There are several studies in the incidence of severe hypoglycaemia in type 1 diabetics already following an insulin treatment, ranging from 62 to 320 episodes per 100 patient-year as compiled in Desouza et al. (2010).

To address conditions such as hypoglycaemia we have developed an agent prototype that monitors blood glucose levels of a diabetic patient as shown in Fig.1. The monitoring knowledge and guidelines required for conditions such as hypoglycaemia, have been specified using a symbolic, computational logic approach combined with temporal reasoning of the kind supported by the Event Calculus of Kowalski and Sergot (1986). This approach is particularly suitable for reasoning about observations according to medical guidelines and has been combined with diagnostic reasoning to provide the patient with suitable recommendations and explanation, even in the light of incomplete information. However, the current monitoring capability cannot cope with information that refers to lifestyle activities of the patient, which are key to diabetes management, especially activities about the patient's physical exercise and diet. Therefore, our *goal* is to provide a run-time monitoring agent that reasons about life threatening situations regarding the diabetic patient based on the perceived events such as glucose readings and user activities. We envisage such an agent to be situated in the patient's mobile phone and directly interact with it to send alerts to the patient.

This paper significantly extends and has the following contributions with respect to our previous work (Kafalı et al., 2014):

- We provide a specification for an activity recognition capability that is integrated within the logic-based agent architecture discussed in Kafalı et al. (2013). The activity recognition capability supports reasoning about complex activities from the recognition of basic events. It relies on the identification of an *activity lifecycle* that treats activities as special temporal fluents that can be started, interrupted, suspended, resumed, or completed over time. Such information is related with a similar lifecycle about the patient's goals, and is amalgamated with a monitoring capability to improve the advice and explanation offered to the patient, as well as corroborating hypotheses about conclusions that require further action. We build a complete implementation of our activity recognition framework, which extends the original prototype and its knowledge representation reported in Kafalı et al. (2014).
- We build an activity generator, a component interesting in its own right, that automatically creates synthetic sensor data in the form of event streams that represent the everyday lifestyle of a type 1 diabetic patient.
- We perform extensive experiments using the generated activity data. We evaluate the recognition capability of our framework using relevant queries for diabetes such as identifying the number of falls/faints occurrences per month, or understanding the glucose trend of the patient during a faint. We evaluate our framework on large datasets under the assumption that makes a time-window approach invalid (e.g. forgetting information is not a reasonable assumption), and report its performance based on the underlying compiled version of the Event Calculus.

The rest of the paper is structured as follows. Section 2 reviews the relevant literature on activity recognition. Section 3 presents our use case, that is used throughout the paper. Section 4 describes the components of our activity recognition framework. Section 5 describes the implementation details of our framework and explains the experimental setting. Section 6 reports the recognition and performance results. Section 7 summarises our work and discusses possible future extensions.

2. RELATED WORK

Activity recognition aims to recognise the behaviour of one or more agents, whether human or artificial, resulting from monitoring a series of observations on the agents' actions and the spatiotemporal conditions of the environment in which these agents are situated, see (Turaga et al., 2008; Avci et al., 2010; Aggarwal and Ryoo, 2011) for recent reviews of the field and related applications. The means by which we obtain observations about the actions of the monitored agent naturally divide activity recognition into two main groups: video-based (Aggarwal and Ryoo, 2011; Turaga et al., 2008), where the task is to recognise a sequence of images with one or more agents performing a certain activity, and sensor-based activity recognition (often called "motion analysis"), which deals with data coming from sensors like accelerometers, gyroscopes and, in general, any readings which could be produced by a mobile or wearable device such as a mobile phone, an activity tracker or a medical sensor. In this work we are concerned with sensor-based activity recognition of a

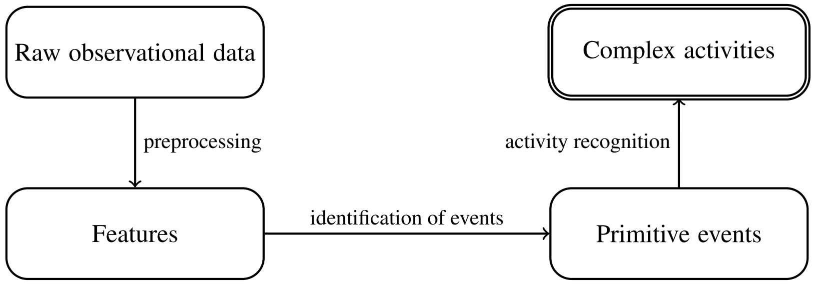


FIGURE 2. General data flow for activity recognition.

single human agent (the user), as opposed to the activities of multiple users/agents (Gu et al., 2009) or the activities of a group seen as a single entity (Gordon et al., 2013).

Activity recognition approaches regarding a single user typically assume the operational flow depicted in the schema of Fig. 2. First, a stream of observational data is received from sensors - possibly mobile - and other sources (e.g. a smartphone or the CGM of a diabetic patient). Second, these raw observations are preprocessed in a standard manner to obtain usable features for the following stages. Then, using these features, computational models recognise a set of low-level primitive events or actions (Turaga et al., 2008). These events for our setting would correspond to the recognition of simple physical actions (e.g. a person walking). Finally, the primitive events together with the context such as historical information and user confirmations are used to recognise more complex activities (e.g. a hypoglycaemic attack for a patient), represented in terms of the events which are captured in the previous level (e.g. a patient stopped walking and then fainted). In our work we mainly focus on the last step to recognise complex activities from primitive ones, represented in Fig. 2 with a double box.

The recognition of complex activities is often formulated using logic-based symbolic reasoning and in some of these cases it is linked to plan recognition (Kautz, 1987), goal recognition (Lesh and Etzioni, 1995) or intention recognition (Sadri, 2012a). Plan recognition refers to the mapping of sequences of atomic actions to high-level plans stored in a plan library, see Krüger et al. (2013); Mirsky et al. (2016); Uzan et al. (2015). As our approach is based on a logic-based formulation too, the similarity with plan recognition lies with our use of domain-specific logic rules that recognise complex activities when specific patterns of actions are observed by the agent that performs the monitoring task. However, our monitoring agent does not rely on the notion of plans and does not map the observed actions to plan instances from an explicit plan library. Rather, it focuses on avoiding unexpected failures by recognizing activities that interfere with the user's high-level goal. Similarly, goal and intention recognition normally focus on the recognition of the high-level goals of actions from observations (Geib et al., 2015; Do et al., 2013). Although the agent-oriented system that we propose considers high-level goals explicitly and links them to activities, it does not perform automatic recognition of goals (or intentions) because one of our system requirements was that these goals should be explicitly specified by the user to trigger the top-level monitoring process performing the recognition tasks.

Activity recognition is often studied using probabilistic techniques to deal with the uncertainty originated from inherently noisy sensors. Good recognition results have been obtained using generative models such as Hidden Markov Models (HMM) (Patterson et al., 2005; van Kasteren et al., 2008) and discriminative models such as Conditional Random Fields (CRF) (Chieu et al., 2006; van Kasteren et al., 2008) for both basic and complex activities. Aggarwal and Ryoo (2011), for example, report very high values in accuracy for basic activities showing performances from 85% to 95%. These models have been extended using hierarchical (Liao et al., 2007; Oliver et al., 2004) and segment (Duong et al., 2009; Truyen et al., 2008) models to handle inter-dependencies between activities in time. However, all these models are supervised models and therefore require labelled data to learn model parameters. In our previous work (Luštrek et al., 2015) we have used supervised models combined with symbolic rules on low-level activity datasets within the COMMODITY_{12} project (Kafalı et al., 2013) to predict high-level activities such as eating and exercise. Still, these activities do not test directly life threatening situations for a patient. In addition, no life threatening situations arose during the clinical trials carried out with real patients using the COMMODITY_{12} system, and thus the scalability of potentially useful functionality in our system could not be tested with real data. As a result, we extend here the logic-based, symbolic approach of COMMODITY_{12} (Kafalı et al., 2013) with complex activity recognition on synthetic data to show how to recognise life threatening situations in a scalable way.

Depending on the methods being used to perform the recognition task, symbolic approaches for complex activity recognition are divided into syntactic and description-based. In syntactic approaches activities are defined as production rules of a grammar, reducing the problem of recognition to the one of parsing. In order to handle uncertainty, stochastic context grammars have often been used (Minnen et al., 2003). Joo and Chellappa (2006) propose a framework for recognition of events using attribute grammars. They represent sequences of events as grammar rules, they assign attributes to each event, while primitive events are represented with terminal symbols. Using this representation, they look for patterns in video sequences that match corresponding rules. Each rule is associated with a probability stating how probable that sequence of events leads to the subject activity. They evaluate their approach with video data from two different domains: casing vehicles in a parking lot and departure of aircrafts. The main limitation of syntactic approaches is in the recognition of concurrent activities. As argued in Ryoo and Aggarwal (2009) syntactic approaches are able to probabilistically recognise hierarchical activities composed of sequential sub-events, but are inherently limited on activities composed of concurrent sub-events. Since syntactic approaches are modeling a high-level activity as a string of atomic-level activities composing them, temporal ordering of atomic-level activities has to be strictly sequential. Our formulation of activity recognition does not suffer from this limitation, as events in our Event Calculus formulation can happen at the same time.

Description-based approaches represent the temporal and spatial structure of activities which they seek to recognise (Nevatia et al., 2003; Hongeng et al., 2004; Francois et al., 2005; Vu et al., 2003). A high-level activity is understood in terms of relationships between simpler, more basic activities (or sub-events) composing the activity. In addition, a time interval maybe associated with an occurring sub-event to specify necessary temporal relationships among sub-events. To specify relationships (sequential, concurrent, and their combinations) and time intervals explicitly, the interval temporal logic predicates discussed in Allen and Ferguson (1997) have been widely adopted by these approaches. The approach presented in this paper is description-based too but instead of Allen's interval logic we use the Event Calculus (Kowalski and Sergot, 1986) to formulate the run-time monitoring required for activity recognition. Our monitoring approach is similar to that of run-time monitoring of Chesani et al. (2013) who use a variant of the Event Calculus called the Reactive Event Calculus ($\mathcal{REC}$). In particular, our treatment of activity and goal lifecycles is similar to their commitment lifecycle. However, we are not interested in modeling agent interactions as they do. Instead, we model the activities and goals of a human user in a software agent who uses these models to recognise the human user's activities.

Artikis et al. (2012) study a Run-Time Event Calculus (RTEC) for recognising composite events at run-time. Although composite events are like complex activities, an activity (composite event) lifecycle as the one we specify is missing. Thus, the difference between our work and RTEC is that our recognition process is more methodological and includes goals for activities as an important requirement for the background knowledge of the recognition process. Another important difference between our work and RTEC is that RTEC focuses on query-time reasoning with time-windows and a forgetting mechanism (Artikis et al., 2015). Instead we focus on update-time reasoning, as in the Cached Event Calculus (CEC) (Chittaro and Montanari, 1996), for large narratives that may cover more than a year of a patient's activities. CEC type of update-time reasoning has been applied in medical applications before (Chittaro et al., 1995) for monitoring but not for activity recognition. Our compiler's implementation is in fact an extended version of a specialised CEC for multi-valued fluents using the weak interpretation of initiates_at/3. According to this interpretation, the update predicates of CEC called propagateRetract/2 and propagateAssert/2, which are the most computationally expensive at update-time, do not have to deal with persistence in the past. Our extension also uses focused forward reasoning from an update event using causes_at/3 definitions, to find out which other events are caused from that event and update the knowledge of the agent with their effects to aid recognition. Any caused (derived) events from a specific update are also cached for further use.

The recognition of physical activities using smartphones and wearable sensors for healthcare is discussed in Kouris and Koutsouris (2012). The emphasis of this study is on the recognition of low-level events using machine learning methods such as decision trees or Bayesian networks, and little is shown about recognising higher-level (complex) activities. As with works in sensor-based human activity recognition (Lara and Labrador, 2013) they deal with recognition of low-level events by trying to find models achieving minimum error. For the specific case of diabetes, not much work has been reported in the literature. A system for monitoring diabetic patients with an activity-recognition module is presented in Helal et al. (2009). While the work describes the architecture of the system and the use of HMMs in the context of a smart home, there is little discussion about the possible activities that could be recognised to aid the lifestyle of a diabetic patient, as in our work.

Han et al. (2012) introduce the notion of "Disease Influenced Activity", which focuses on monitoring uncommon patterns for diabetes, such as "frequent drinking", which are presented offline to a doctor (see Shoaib et al. (2015) for a review of offline activity recognition approaches using embedded sensors on mobile phones). The fundamental difference between "frequent drinking" and the "life threatening" activities we study in the paper is as follows. The former is about activities that influence the management of the disease in the medium/longer term and need to be reported to the doctors who follow the patient for information about the patient's lifestyle. On the other hand, the latter is about activities that need to be dealt with immediately and alert the doctors now. In addition, in our work the focus is to use a mobile phone to recognise life threatening situations online to assist the patient, however, the observations can be shared with doctors offline, if necessary.

With the increasing popularity of lightweight and affordable wearable sensors the continuous monitoring of patients with chronic diseases has become easier. In this context the role of activity recognition has become to detect abnormal situations and alert the patients to prevent life threatening situations (see Mukhopadhyay (2015) for a survey on wearable sensors). However, the engagement of patients in such wearable data collection has been found to be challenging. Our approach to make activity recognition available on mobile phones is an attempt to support more patient-centered approaches by aiding usability in care, and secure handling of patient information with familiar devices (Chiauzzi et al., 2015).

Approaches for the detection of suspicious behaviour using video and audio data streams (Arroyo et al., 2015; Elhamod and Levine, 2013) complement our recognition framework in certain situations. Although monitoring a patient for life threatening situations does not provide a controlled environment in comparison to a video surveillence problem (e.g. people going into a shopping mall or using public transport hubs), we can utilise available components of a smart home (Jakkula and Cook, 2011) or a smart city infrastructure to

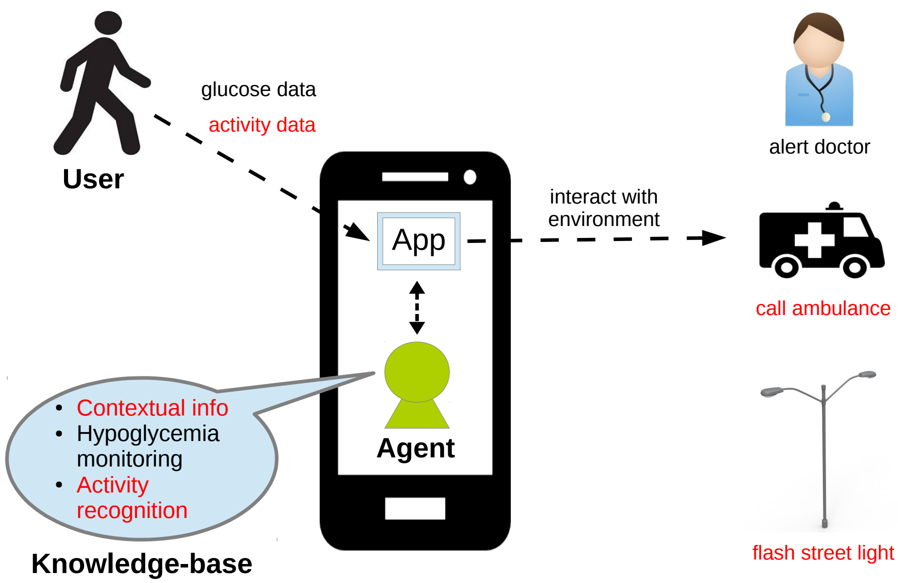


FIGURE 3. Diabetes monitoring and management in our framework. Red components show the additional features of the continuous monitoring agent extended from the CGM agent in COMMODITY$_{12}$ (Kafalı et al., 2013).

improve recognition accuracy (e.g. the agent accesses a security camera feed to confirm a faint event). Another relevant area of research is detecting anomalous behaviours in social networks (Viswanath et al., 2014), which employs unsupervised learning techniques to detect and prevent cybersecurity attacks using fake or compromised user accounts. Our approach is similar in that we do not require labeled data to recognise faint events.

3. USE CASE: SAFETY-ENHANCED SMART STREET

We present a scenario from healthcare regarding the everyday lifestyle of diabetic patients, in order to demonstrate the significance of activity recognition in such a setting.

John, a *type 1 diabetic*, is returning home after having spent an evening to the movies with friends. The bus that he took to go home does not reach John's street directly, so John needs to walk back to his place. Once he alights from the bus John feels a bit weak. As a precaution he starts a mobile phone app that allows him to set high-level goals, in this case *walk home*, to be monitored until they are achieved. The app does so by capitalising on knowledge it has about when such goals are considered achieved, in this case by comparing John's home address and his current GPS location coordinates. After John specified that he wants to walk home, the app estimates that the walk will take him approximately 20 minutes. Halfway, however, John receives an alert informing him that the content of glucose in his blood is abnormally low (a hypoglyceamia medical emergency). John does not have enough time to respond to this alert as he passes out and falls on the pavement. Immediately after John falls on the pavement, his doctor and family are informed, an ambulance is called, and the nearest street light starts flashing to attract attention of passers-by and help the ambulance locate John.

To support such a scenario we assume that John's mobile app is developed as a software agent that monitors John's glucose with an insulin pump and recognises John's activities in relation to his diabetes. The insulin pump is a device that measures blood glucose, holds an insulin cartridge and can deliver a continuous flow (basal rate) of insulin to the body at the press of button. In regular intervals, it can communicate to the mobile app the patient's glucose measurements, so that the agent can detect abnormal glucose readings.

The scenario above requires that if the glucose level is dangerously low, the agent must take a number of important steps. Immediately after sending the hypoglycaemia alert, the agent must also ask John via the app's display whether he feels well. If John does not respond because he fainted, this can be recognised because the agent can observe that John fell while suffering a hypoglycaemic attack. As a result, the agent must alert first John's doctor, then John's family and an ambulance giving John's location. We also assume a neighbourhood e-infrastructure of the kind envisaged in Connected Communities (e.g. Mamdani et al. (1999); Stathis et al. (2006)) and more recently in Smart Cities (e.g. Schaffers et al. (2011)). Using such an e-infrastructure, the agent can observe the closest street light, also represented electronically as a software agent, requesting it to flash about John's medical emergency.

4. THE ACTIVITY RECOGNITION FRAMEWORK

In this section, we present various elements of our activity recognition framework.

4.1. Architecture

Fig. 3 shows how our agent framework, presented in the introduction, is extended with activity recognition to support the smart street scenario. We use dark font to represent the currently supported features of the monitoring agent within the personal health system presented in Fig. 1. We extend this original framework with a new set of features relevant to complex activity recognition. The agent is situated in the smart phone of the user and interacts with the application that receives input such as glucose and activity data from the sensors on the user. The agent's knowledge-base is also extended with logic rules regarding activity recognition to process activity data (see Sections 4.2 and 5.1) as well as contextual information about the user's intentions (e.g. the user's current goal). The agent can also interact with the user's surroundings – in an emergency it calls an ambulance, flashes the street lights to attract attention and alerts the user's doctor.

4.2. Recognising Lifecycle Transitions of Activities with the Event Calculus

We are now ready to describe our activity recognition framework. In this framework an activity is understood as a parameterised template whose parameters consist of a label naming the activity, a set of participants co-involved in the carrying out of the activity and a goal revealing the desired outcome of the participants participating in it. The framework identifies an activity lifecycle that presupposes the occurrence of primitive events (e.g. walks, stands, lies) representing the input arriving from a low-level recognition system (see Fig.2). Then the occurrence of primitive events treats activities as temporal fluents that can be started, interrupted, suspended, resumed, or completed in time. The framework is driven by an additional template for activity goals and their associated lifecycle, similar to that of activities. Both lifecycles are presented in Fig.4.

To reason about the evolution of activities and the effects of events we use the Event Calculus of Kowalski and Sergot (1986). Table 1 summarises the ontology of the domain-independent axioms we employ by selecting multi-valued fluents as in Artikis et al. (2009) to represent state properties including activities and goals. We further extend the ontology of the Event Calculus with a $\mathsf{causes\_at/3}$ predicate in order to represent explicitly when an event causes another event to happen at a specific time. We will see later that this allows us

TABLE 1. Ontology of our version of the Event Calculus.

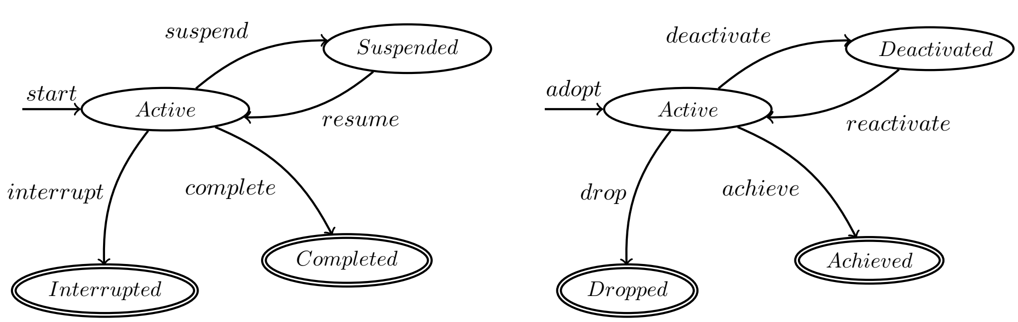
Predicate	**Description**
happens_at(E, T)	Event E happens at time T
holds_at(F=V, T)	Fluent F has value V at time T
holds_for(F=V, [Ts,Te])	Fluent F continuously has value V from time Ts to time Te
broken_during(F=V, [Ts,Te])	Fluent F has changed value V from time Ts to time Te
initiates_at(E, F=V, T)	Event E initiates value V for fluent F at time T
terminates_at(E, F=V, T)	Event E terminates value V for fluent F at time T
causes_at(E1, E2, T)	The occurrence of event E1 causes event E2 to happen at T

to rewrite rules using happens_at/2 in both the head of a rule and in its body, thus saving us from performing expensive forward reasoning when occurrence of certain events cause others to happen. We will elaborate on this point further in Section 5.3, where we will discuss the implementation techniques that we will use for our system.

On top of the domain-independent axioms, our framework consists of the following additional components:

- an activity theory that follows the activity lifecycle;
- a goal theory that follows the goal lifecycle;
- a domain model that describes the recognition domain;
- an event narrative that contains the events that happened in the system.

We start with the generic components of the event recognition framework, i.e. the activity theory and the goal theory (see Section 5.1 for the domain model and the event narrative). Fig. 4 describes the lifecycle of an activity (a) and a goal (b). The recognition of activities is driven by the goals of the user, which we represent as a modification of the goal lifecycle presented in van Riemsdijk et al. (2008) for our purposes. An activity is first activated due to a goal being adopted by the user and a low-level event happening to start the activity. While the activity is being performed, if the user's goal changes, then the activity is no longer required (e.g. the goal is *dropped*), then the activity is *interrupted*. If the goal remains, but another goal supersedes it temporarily (e.g. the goal is *deactivated*), then the activity is *suspended*. When the user reactivates the goal again, the activity is resumed. The activity completes successfully when the user achieves the goal, in which case the activity is *completed*.



(a) Activity lifecycle (b) Goal lifecycle

FIGURE 4. Lifecycle of an activity and a goal. Double ellipses represent terminal states.

Listing 1 presents the Event Calculus axioms specifying the domain independent activity lifecycle. Note that A is a term representing an action. Terms that start with capital letters represent variables as in Prolog notation. Lines 1–10 describe how the recognised events initiate different values for the activity fluents; termination of these fluents are handled automatically by a generic terminates_at/2 definition, see Artikis et al. (2009) (axiom 19).

```
1  initiates_at(start(A), A=active, T) :-
2    happens_at(start(A), T).
3  initiates_at(suspend(A), A=suspended, T):-
4    happens_at(suspend(A), T).
5  initiates_at(resume(A), A=active, T):-
6    happens_at(resume(A), T).
7  initiates_at(interrupt(A), A=interrupted, T):-
8    happens_at(interrupt(A), T).
9  initiates_at(complete(A), A=completed, T):-
10   happens_at(complete(A), T).
```

LISTING 1. Domain independent activity theory.

Similar to the activity lifecycle, Listing 2 presents the Event Calculus axioms for the goal lifecycle, where G represents a goal. Lines 1–10 describe how the goal events initiate different values for the goal fluents.

```
1  initiates_at(adopt(G), G=active, T):-
2    happens_at(adopt(G), T).
3  initiates_at(deactivate(G), G=deactivated, T):-
4    happens_at(deactivate(G), T).
5  initiates_at(reactivate(G), G=active, T):-
6    happens_at(reactivate(G), T).
7  initiates_at(drop(G), G=dropped, T):-
8    happens_at(drop(G), T).
9  initiates_at(achieve(G), G=achieved, T):-
10   happens_at(achieve(G), T).
```

LISTING 2. Domain independent goal theory.

We show next how to develop the domain dependent part of our framework in order to support the activity recognition we envisage for our scenario. We represent an activity fluent as activity(Name, Participants, Goal)=State. The Name is an atom (e.g. walking), the Participants is either a list of atomic identifiers (e.g. [john, peter] or a single such identifier (e.g. john), and Goal is the name of a goal that specifies what the activity is seeking to achieve (e.g. at_home) with the possibility of a *null* value. The State represents the current state of the activity, which is drawn from the set of possible values active, suspended, interrupted and completed (see Fig. 4(a)). Similarly, we represent a goal fluent as goal(Name, Participants)=State. The State represents the current state of the goal, which is drawn from the set of possible values active, deactivated, dropped and achieved (see Fig. 4(b)).

4.3. Domain Dependent Definitions for Activities and Goals

We show next how to define domain-dependent clauses to specify the lifecyles of activities and goals for the specific low-level events of an application. Listings 3 and 4 assume the person has the goal of going home (G=at_home). Listing 3 shows an extract of the domain dependent activity theory exemplified, in part, by the activity of walking. The starting of this activity is caused when a low-level event walks(P) happens (stating that the participant P walks, see lines 1-5). We assume that the low-level activity recognition module will not send us more low-level walks(P) events, only when it recognises that walking has stopped and something else has happened. When a new (different) event is recognised by the low-level module, it will be communicated to the high-level one, which will in turn cause to suspend the current activity. Note that the \+ symbol denotes negation as failure (Clark, 1987). Lines (7-10) show how when the low-level event stands(P) happens (stating that the participant P stands) causes walking to be suspended. The walking activity is resumed (becomes *active* again) when a low-level walks(P) event happens (Lines 12-15). An activity is interrupted when that activity's goal is dropped (Lines 17-19), and, an activity is *completed* when that activity's goal has been achieved (Lines 21-22), but refer to Listing 4 for the detailed conditions of this happening.

```
1  causes_at(walks(P),  start(activity(walking, P, G)), T):-
2    happens_at(walks(P), T),
3    holds_at(goal(G, P)=active, T),
4    \+ holds_at(activity(walking, P, G)=active, T),
5    \+ holds_at(activity(walking, P, G)=suspended, T).
6  ...
7  causes_at(stands(P), suspend(activity(walking, P, G)), T):-
8    happens_at(stands(P), T),
9    holds_at(goal(G, P)=active, T),
10   holds_at(activity(walking, P, G)=active, T).
11 ...
12 causes_at(walks(P), resume(activity(walking, P, G)), T):-
13   happens_at(walks(P), T),
14   holds_at(goal(G, P)=active, T),
15   holds_at(activity(walking, P, G)=suspended, T).
16 ...
17 causes_at(drop(G, P), interrupt(activity(A, P, G)), T):-
18   happens_at(drop(G, P), T),
19   holds_at(activity(A, P, G)=active, T).

21 causes_at(achieve(goal(G, P)), complete(activity(A, P, G)), T):-
22   happens_at(achieve(goal(G, P)), T).
```

LISTING 3. An example of domain dependent activity theory.

Similarly, Listing 4 shows an extract of a domain dependent goal theory exemplified, in part, by assuming that the diabetic user will be specifying the goal a graphical user interface (GUI) of the mobile phone application. In this domain, actions of the user at the GUI are interpreted as internal events that cause the adoption of a new goal (Lines 1–4) or the deactivation / reactivation / dropping of an existing goal (Lines 6–8, 10–12, and 14–16 respectively). Finally, the achievement of the goal is specified by the person arriving at the destination specified in the goal (Lines 18–22).

```
causes_at(adopt_goal_fromGUI(P, G), adopt(goal(G, P)), T):-
  happens_at(adopt_goal_fromGUI(P, G), T),
  \+ holds_at(goal(G, P)=active, T),
  \+ holds_at(goal(G, P)=deactivated, T).

causes_at(deactivate_goal_fromGUI(P, G), deactivate(goal(G, P)), T):-
  happens_at(deactivate_goal_fromGUI(P, G), T),
  holds_at(goal(G, P)=active, T).

causes_at(reactivate_goal_fromGUI(P, G), reactivate(goal(G, P)), T):-
  happens_at(reactivate_goal_fromGUI(P, G), T),
  holds_at(goal(G, P)=deactivated, T).

causes_at(drop_goal_fromGUI(P, G), drop(goal(G, P)), T):-
  happens_at(drop_goal_fromGUI(P, G), T),
  holds_at(goal(G, P)=active, T).

causes_at(arrived(P, L), achieve(goal(G, P)), T):-
  happens_at(arrived(P, L), T),
  holds_at(goal(G, P)=active, T),
  holds_at(destination_of(G)=L, T),
  holds_at(activity(A, P, G)=active, T).
```

LISTING 4. An example of domain dependent goal theory.

5. EXPERIMENTS

In this section, we (i) formalise the use case described in Section 3 in the Event Calculus, (ii) describe an activity generator to produce narratives relevant to the use case, (iii) propose a process for incremental compilation of such narratives in the Event Calculus, and (iv) describe our experimental setup regarding the use case.

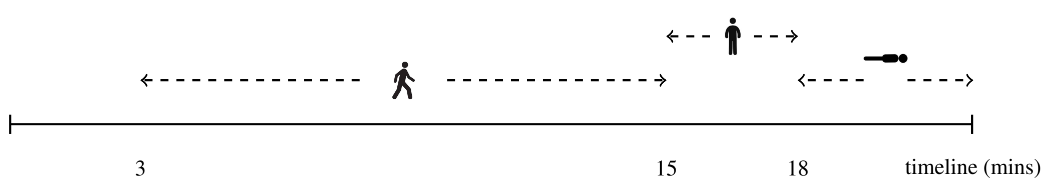


FIGURE 5. John's activities.

5.1. Implementation of the Use Case

We now focus on the scenario described in Section 3, and implement the domain specific predicates of Event Calculus for recognising the falling and fainting of a person[1]. Let us first review the primitive events that lead to John falling on the street due to a hypoglycaemia episode. Fig. 5 shows the timeline of John's activities after he gets off the bus and heads home. We capture the temporal intervals of such activities using the predicate holds_for/2, implemented in our Event Calculus representation. It represents the validity period for activities that are in active or suspended state. This is shown in Listing 5. John stops walking

[1]The complete implementation can be found at https://bitbucket.org/dice_rhul/mvfcec.

at time 15. This initiates a transition in the activity lifecycle: standing becomes active and walking becomes suspended in the next time point (16). Similarly, when John stops standing at time 18, this initiates a transition in the activity lifecycle: lying becomes active and standing becomes suspended in the next time point (19).

```
happens_at(adopt_goal_fromGUI(john, at_home), 1).
happens_at(walks(john), 3).
happens_at(stands(john), 16).
happens_at(lies(john), 19).

holds_for(activity(walking, john, at_home)=active, [3, 15]).
holds_for(activity(standing, john, null)=active, [16, 18]).
holds_for(activity(lying, john, null)=active, [19, infPlus]).

holds_for(activity(walking, john, at_home)=suspended, [16, infPlus]).
holds_for(activity(standing, john, null)=suspended, [19, infPlus]).
```

LISTING 5. Recognition of intervals for primitive activities.

Using this knowledge only, we can recognise if someone is falling. Listing 6 describes a basic approach for the recognition of the composite event *falls*. The person must go from walking to standing, and then to lying in a short period of time in order to be recognised as a fall event. Note that this rule does not take into account the activity theory described in Section 4.2, and thus requires explicit temporal interval reasoning (i.e. the predicate immediately_before/2) to check the order of activities.

```
happens_at(falls(Person), T):-
  happens_at(lies(Person), T),
  holds_for(walking(Person)=true, [_, T1]),
  holds_for(standing(Person)=true, [T2, T3]),
  immediately_before(T1, T2),
  immediately_before(T3, T).

immediately_before(T1, T2):-
    T is T2 - T1,
    lt(T,2).
```

LISTING 6. Recognition of a fall event without the activity theory.

Listing 7 improves the previous rule with the use of our activity framework. Here, since the states of the activities are handled by the activity theory, the rule does not need to check explicitly the validity periods of the walking and standing fluents as previously done with the immediately_before/2 predicate shown in Listing 6. Because we use activity lifecycles to develop domain specific rules linking the goals of a person, these domain specific rules can use the values of activity fluents (e.g. suspended, active, completed) and the events that happen at a time T to recognise which new events are caused. For example, if the low level event that the person is lying down has been recognised at time T, and at that time standing is active for the goal to go home, and walking is suspended for that goal, then the fact that he is now lying causes the recognition of the event that the person has fallen. If we did not have the activity lifecycles with the goal, the search for previous activities (i.e. walking) is not guided. As a result, we need to query for previous times T_1, T_2 and T_3 (as described in Listing 6), which are not instantiated at query time and thus make the system less efficient in searching unguided for the previous activity as there could be other occurrences in which walking could have been active but without our framework we cannot express the fact that these could have been related to other goals and not the current one.

```
causes_at(lies(Person), falls(Person), T):-
  happens_at(lies(Person), T),
  holds_at(activity(standing, Person, G)=active, T),
  holds_at(activity(walking, Person, G)=suspended, T).
```

LISTING 7. Recognition of a fall event using the activity theory.

In order to recognise that John has fainted, we need additional knowledge about the environment as well as the intentions of John. Listing 8 describes this domain knowledge relevant to our scenario. John's goal is to walk home after watching the movie. As he starts walking home after he gets off the bus, he receives a hypoglyceamia alert and stops to look at his smartphone. Unfortunately, he falls down soon after checking the alert. The agent running on his smartphone asks for John's status immediately after he falls.

```
happens_at(adopt_goal_fromGUI(john, at_home), 1).
happens_at(measurement(john, glucose, 2.8), 14).
happens_at(requests(john, confirm_status), 20).
```

LISTING 8. Contextual information significant to recognising event interruption.

Now we can combine this knowledge together with the formalisation of the fall event to conclude that John has fainted. In order to recognise that walking is interrupted (by an emergency) rather than just suspended for a period of time, we need additional contextual information as well as the fact that John has fallen. More specifically, the agent distinguishes fainting from falling if the following happens:

- the agent has sent an alert to John following a hypoglycaemia before he fell,
- the agent has asked John to confirm his status soon after he fell, and it has not received a response,
- John has a goal that has not been achieved yet.

Definition 1 formalizes a faint event according to the above description.

Definition 1: A faint event e_f occurs at time t_f in narrative N=$\{\langle e_1, t_1\rangle, \ldots, \langle e_n, t_n\rangle\}$ if and only if

- $\exists\langle e_i, t_i\rangle, \langle e_j, t_j\rangle \in N$: e_i is a hypoglyceamia alert sent by the agent, e_j is a fall of the patient, and $t_i < t_j < t_f$;
- $\exists\langle e_k, t_k\rangle, \langle e_l, t_l\rangle \in N$: e_k is a fall of the patient, e_l is a status confirmation request sent by the agent, and $t_k < t_l < t_f$;
- $\nexists\langle e_m, t_m\rangle \in N$: e_m is a response from the patient to the status confirmation request and $t_i < t_m < t_f$.

Listing 9 describes the rule for recognising faint events. We capture fainting as a special case of the fall event (e.g. the interruption of walking).

```
1 causes_at(no_response(Person, WaitingTime), faints(Person), T):-
2   happens_at(no_response(Person, WaitingTime), T),
3   happens_at(raises_alert(Person, hypoglycaemia), T1),
4   happens_at(falls(Person), T2),
5   lt(T1, T2),
6   happens_at(requests(Person, confirm_status), T3),
7   lt(T2, T3),
8   T4 is T3 + WaitingTime,
9   holds_at(goal(Goal, Person)=active, T),
10  ge(T, T4).
```

LISTING 9. Recognition of a faint event.

Theorem 1: Soundness: Listing 9 correctly recognises a faint event.

Proof. We prove Theorem 1 by contradiction. Assume Listing 9 recognises an event e_f as faint, and e_f is not a faint event (Definition 1). Then, either of the following must hold:

- $\nexists\langle e_i, t_i\rangle, \langle e_j, t_j\rangle \in N$: e_i is a hypoglycaemia alert sent by the agent, e_j is a fall of the patient, and $t_i < t_j < t_f$. Lines 3–5 of Listing 9 ensure that this is not the case. Therefore, this is a contradiction.
- $\nexists\langle e_k, t_k\rangle, \langle e_l, t_l\rangle \in N$: e_k is a fall of the patient, e_l is a status confirmation request sent by the agent, and $t_k < t_l < t_f$. Lines 4 and 6–7 of Listing 9 ensure that this is not the case. Therefore, this is a contradiction.
- $\exists\langle e_m, t_m\rangle \in N$: e_m is a response from the patient to the status confirmation request and $t_m < t_f$. Lines 1–2, 8, and 10 of Listing 9 ensure that this is not the case. Therefore, this is a contradiction.

□

After the agent detects there is something wrong with John, it has to take appropriate action to make sure John is safe. Listing 10 describes the events that connect the agent with the environment. It can alert his doctor and call an ambulance as well as interacting with the street lights (provided a suitable infrastructure).

```
alert(doctor). %via the smartphone
alert(ambulance). %via the smartphone
alert(street_light). %via the smart city infrastructure
```

LISTING 10. Ambient assist during a faint event.

5.2. Activity Generator

In order to test both the flexibility and the efficiency of our approach we have developed an event generator for the diabetes scenario proposed in section 3. Such a generator consists of a discrete event simulation algorithm (Banks et al., 2010; Buss, 1996), and outputs a sequence of events (i.e. an event narrative) coming from the agent's sensors with a set of "days" (one file per day) where one person comes back home walking from work. An example narrative is shown below in Listing 11. Fig. 6 shows the corresponding event graph for the generator. The generator is not designed to produce data similar to the real one but instead, to create a dataset large enough to prove that our approach is able to deal with an important volume of events, and to prove that it properly detects the faint ones, even when some randomness is introduced. Thus, even when faint events should be very rare in

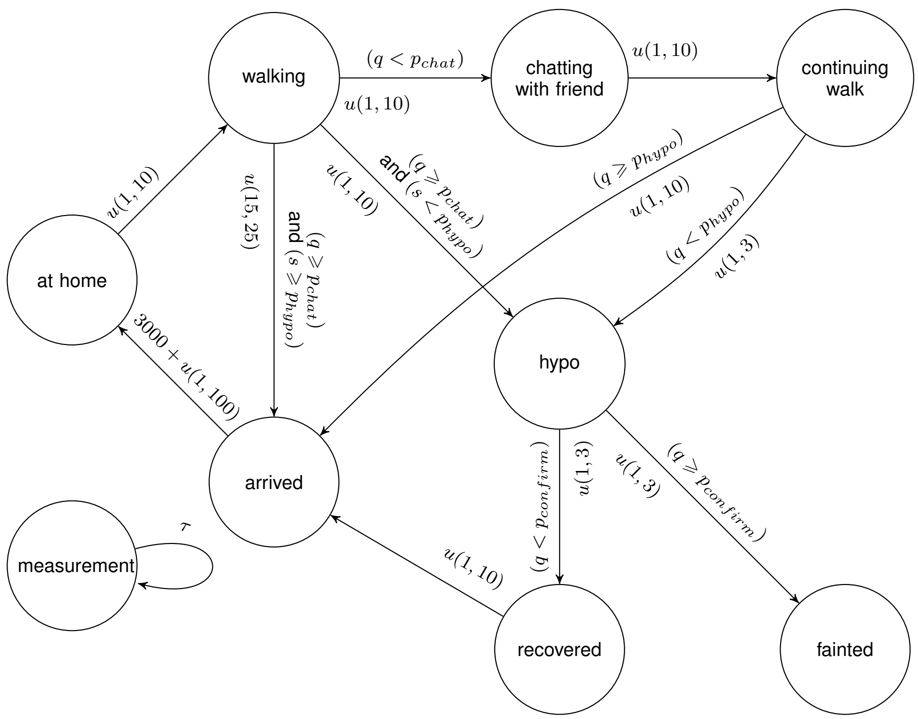


FIGURE 6. Event graph for the synthetic data generator where nodes represent different events and arrows possible transitions between them. A transition can only happen if the condition stated between parentheses in the edge holds (some transitions are 'unconditional', meaning that the following event always holds). The time for the next event is computed as the current time plus a (possibly random) quantity, placed at the beginning of the node. We denote by $u(a,b)$ a random integer number between a and b, and q and s are random real numbers in $[0,1]$. When one event can have different events as the next one, only one is chosen (conditions are mutually exclusive), therefore only one single value for q and s are computed before each transition. Finally, the parameters of the model are $p_{hypo} = 0.05$, $p_{confirm} = 0.5$, $p_{chat} = 0.25$ and the measurement period, τ is adjusted to produce 500 measurements per day. The simulation starts with two events in the queue, a measurement event and at_home, both happening at time zero.

a real environment, we have introduced certain parameters that overestimate the probability of these, therefore producing many of them that can be used as a validation of our method.

Before the walk, an event is generated (possibly from the GUI of a mobile phone application) setting the goal to go home. The rest of events happening are randomly generated according to several parameters (probability distributions) that can be specified by the user.

Events are processed sorted by their $time$ attribute, in which they are supposed to happen. Therefore, the list of events is a priority queue (ordered by time), and every event which is processed generates future events which are introduced in the queue accordingly. This general scheme is summarised in Algorithm 1 which is shown in the Appendix. Each event is processed and the Prolog narrative corresponding to that event is generated (this is summarised in Algorithm 2 in the Appendix). After that processing, from that current event the next event is generated based on several random conditions (this procedure is shown in Algorithm 3 in the Appendix).

Following the set of generated events, the person may (or may not) stop to talk to a friend, may suffer (or not) a hypoglycaemia and, in that case, if the person does not recover, the sequence of events corresponding to a faint episode would be produced. In parallel, the events corresponding to glucose measurements are periodically generated.

```
initially(location_of(john)=busStop).
initially(glucose(john)=3.8).

happens_at(adopt_goal_fromGUI(john, at_home), 1).
happens_at(walks(john), 3).
happens_at(measurement(john, glucose, 3.3), 3).
happens_at(stands(john), 4).
happens_at(measurement(john, glucose, 3.1), 4).
happens_at(walks(john), 5).
happens_at(measurement(john, glucose, 2.8), 5).
happens_at(stands(john), 6).
happens_at(raises_alert(john, hypoglyceamia), 6).
happens_at(lies(john), 7).
happens_at(requests(john, confirm_status), 8).
```

LISTING 11. Narrative of events that leads to fainting of John.

Listing 11 describes a short narrative of events created by the activity generator, that leads to fainting of John. The CGM device takes measurements as John starts his walk from the bus stop towards his house. Note that John's blood glucose gradually drops and reaches a critical value, after which John eventually falls down on the pavement. The last event on the narrative is the message John receives from the agent on his smartphone. John faints before responding to the message. Note that the activity generator does not necessarily reflect real-life situations. This is intentional as there would be very few (if not none) hypoglycaemia related faint incidents in real life. For the purposes of experimentation and validation of our approach, we generate more than usual faint incidents.

5.3. Incremental Compilation of sensor data to EC Narratives

We have developed an update-time reasoning component that processes streams of events arriving to the agent from its sensors. Such events are processed incrementally by calling the updates_at/2 clauses shown in 12 (Lines 1–4). An update with a single event (Lines 6–9) first applies the effects of the event to the knowledge-base and then processes the updates that are caused by the event.

```
1 updates_at([], _).
2 updates_at([Event|Events], T):-
3   update_at(Event, T),
4   updates_at(Events, T).

6 update_at(Event, T) :-
7   effects_at(Event, T),
8   findall(CausedE, causes_at(Event, CausedE, T), CausedEs),
9   updates_at(CausedEs, T).
```

LISTING 12. Processing a list of event updates.

The effects_at/2 predicate essentially asserts the event in the knowledge-base and then maintains a compiled version of maximally validity intervals (MVIs) as in Chittaro and Montanari (1996) but for multi-valued fluents. We represent a multi-valued fluent p with variables X_1, X_2, ..., X_n and value V for an MVI with starting time T_s and ending time T_e

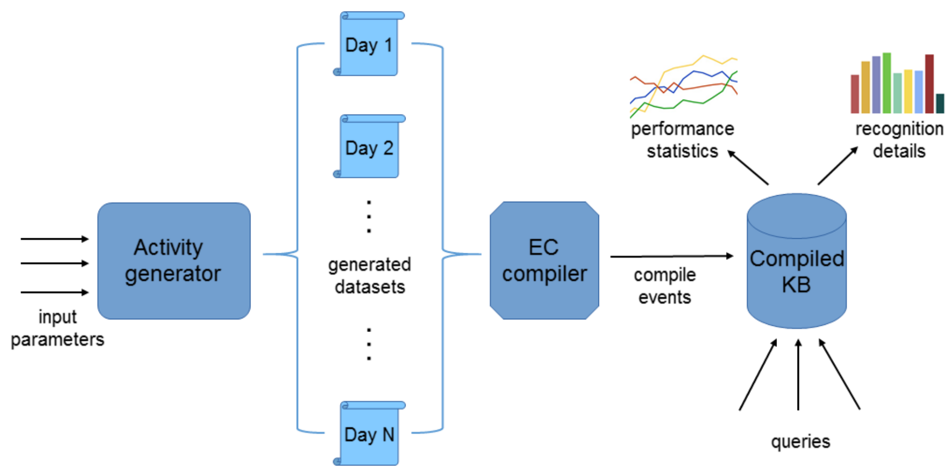


FIGURE 7. Experimental design.

as

holds_for(p(X_1, X_2, ..., X_n) = V, [T_s, T_e])

but we compile it at run-time and store it as:

p_holds_for(p(X_1, X_2, ..., X_n) = V, [T_s, T_e]).

This way of storing MVIs is more efficient as it queries a more specific predicate starting with the name of p rather than the generic holds_for/2. However, as we hide these details from the user of the framework, the query mechanism for holds_for/2 needs to be changed to take into account these compilation details. In this way the compiled version of the MVI is accessed each time a holds_for/2 query is posed to the system.

Once the effects of an event have been applied to the state of the knowledge-base, the events caused by these effects are also added via updates and cached. We find these events straightforwardly using forward reasoning with causes_at/3 definitions and we avoid the unguided forward reasoning that would have been necessary if we used happens_at/2 rules. This point further justifies why we extended with causes_at/3 the original ontology of the Event Calculus in our framework.

5.4. Experimental Setup

We use the activity generator to create and represent the everyday lifestyle activities of a type 1 diabetic patient, John. In total, we create 365 separate narratives with 500 approximately events on average. This represents a time span over a year of John's activities. We configure the activity generator such that approximately 10% of the days include a 'faint' event that John faces. We conduct the following two sets of experiments.

Performance statistics We test the performance of our activity-theory on the combined dataset of the 365 individual day-long datasets. For this purpose, we use the incremental compilation technique as described in Section 5.3. We run different queries for the 'falls' and ‘faints’ events described in Listing 6 (without the activity theory), Listing 7 (using the activity theory), and Listing 9.

Recognition details We provide a set of queries that are supported by our framework to demonstrate various details of recognizing ‘faints’ events. The queries can be used for two purposes: (i) to get general information about a patient such as identifying the number of faint occurrences per month, and (ii) to focus on the details of a specific faint occurrence (e.g. to understand the glucose trend of the patient before and after a faint).

Fig. 7 summarizes our design for running the experiments. First, the activity generator is run by specifying its input parameters such as number of events per day, glucose measurement frequency, etc. Based on the given parameters, it generates a set of datasets, each representing a day of the patient’s activities. These datasets are then fed into the EC compiler, which compiles the events and creates the compiled knowledge-base. During the compilation stage, we collect performance statistics regarding the compilation of events (update time). After the compilation of the whole data is finished, we evaluate the run time performance by running various findall/3 and holds_at/2 queries. We also run queries to identify interesting details regarding the faint occurrences.

6. RESULTS

In this section, first we show the performance results regarding our incremental compilation process for Event Calculus narratives including query times, and then we present sample queries that our framework supports regarding the activities of diabetic patients.

6.1. Performance Evaluation

We will first provide a benchmark on the update time. Using the generator presented in section 5.2, we built a dataset for one single person, for a whole year (365 days) using the default parameters. All benchmarks were run on a computer with an Intel Core i7 2.6GHz processor and 16GB RAM, running Mac OS version 10.10.5 and SWI Prolog Version 7.3.6.

We first show a benchmark on the update time: we start compiling the narrative for the first day and we run updates on our dataset adding the rest of the days one by one until the end of the year. Fig. 8 shows the individual update times for each new day, i.e. the time it takes for compiling the events of day N given that all previous events from days 1 . . . N-1 are compiled in the knowledge-base. The results show that the update time is approximately linear with respect to the number of days already added onto the knowledge-base. The last day of a whole year, where the knowledge-base already contains more than $175,000$ events, can be compiled in less than three minutes, which shows that our approach is efficient, even for high-volume datasets. Since our approach is intended for run-time monitoring, this is a significant result. For example, if we had been monitoring the activities of John for 350 days, each new event in day 351 would take less than one third of a second to compile, which enables our monitoring agent to detect life threatening situations in real-time.

In order to test the performance of the query time in our approach we will show different experiments. First of all, we will run a query to find all “fall” and “faint” events of the patient for a whole year. These two queries are shown in Listing 13. Since these two queries take a really small amount of time, we have repeated them 100 of times and gathered the minimum, maximum and average query time for each one. The query time distributions are shown in Fig. 9(a) and Fig.9(b); fall events are found in less than 0.1 ms, even for the worst case, with an average query time of 0.033 ms. The query for faint events, although more complicated, is still very efficient: they can be queried in less than 20 ms, with an average of 0.2 ms. This

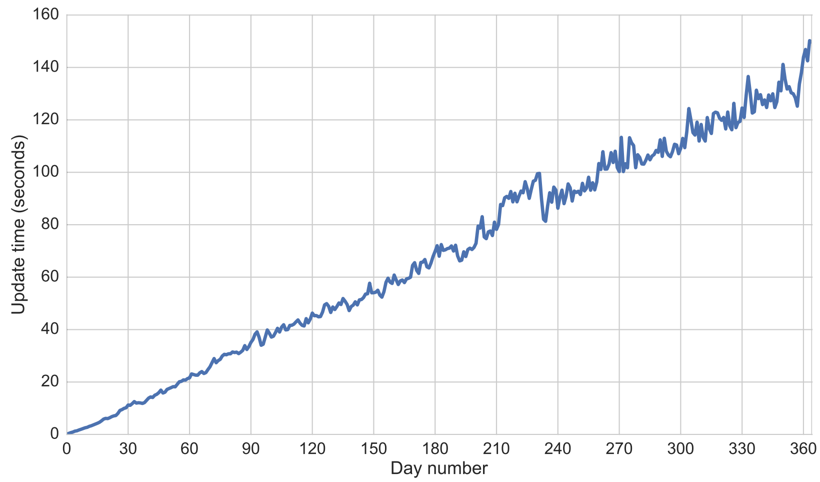


FIGURE 8. Benchmark on the update time.

shows that our approach provides query time which run virtually in real time, our narrative was based on 184,448 events that have happened over the period of a year.

```
findall(T, happens_at(faints(P), T), Ts).
findall(T, happens_at(falls(P), T), Ts).
```

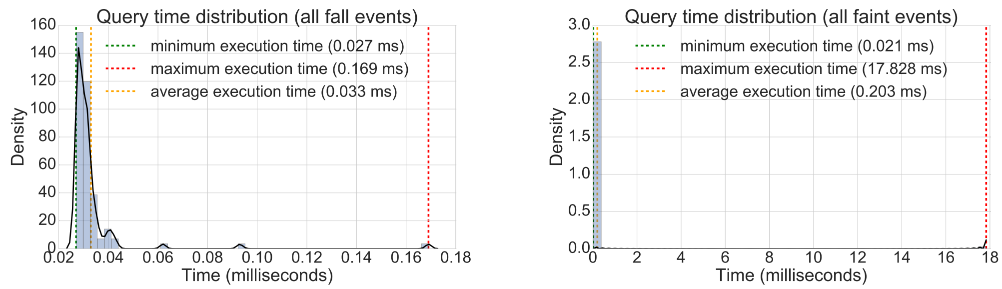
LISTING 13. Finding all fall and faint events.

We have run benchmarks on glucose-related queries, as well. First of all, for each day of the year, we have queried the glucose values of the person in the same dataset. Each query is run 100 times as in the previous case and, in figure 10(a). Query time is very efficient, with a maximum run time of 70 ms for the worst case, and averages of around 10 ms. We show the same experiment in a monthly basis, in figure 10(b), with query times in the same order of magnitude. This clearly shows that, for a use case involving complicated queries over the glucose values the runtime is almost immediate.

6.2. Supported Queries

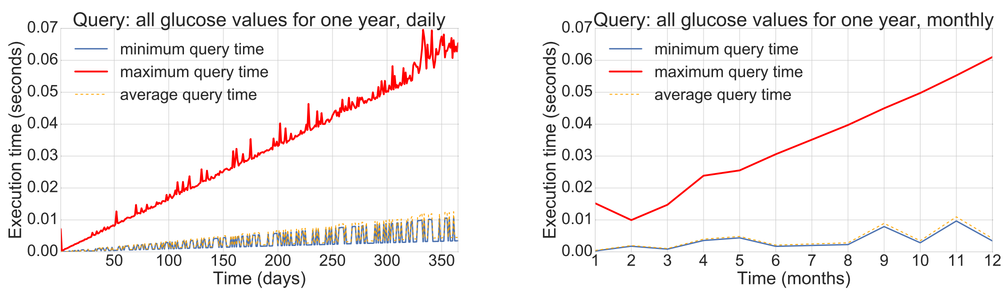
We provide a sample set of queries to demonstrate the sort of information that can be inferred from our activity recognition framework. While our framework supports these queries on the compiled datasets in order to gather details on 'faints' events, the range of possible queries is not limited to the following. A designer may construct other queries depending on the end users' needs. We assume the designer has familiarity with declarative programming. But, the end user would be given an interface to produce customised queries, e.g. daily, monthly, or yearly reports of fainting.

Faint frequency shows the number of faint occurrences per month. This is useful to identify whether the patient has a particular trend during the year. For example, we can answer questions such as 'Are all the faint occurrences uniformly distributed?' or 'Are they clustered in a short period of time, e.g. one month?' Listing 14 lists the query to find all faint occurrences per month. The external user query abstracts away all time-related details of EC, and only requires the user to enter the subject month, e.g. "january". The internal computation of months with respect to time points is performed by the EC reasoner. Each



(a) Fall events (b) Faint events

FIGURE 9. Execution benchmark for 'findall/3' queries.



(a) Each day (b) Each month

FIGURE 10. Execution benchmark to find all glucose measurements.

day has an interval of 2500 timepoints, and starts with where it has left from the previous day. Thus, a month on average has 75000 timepoints.

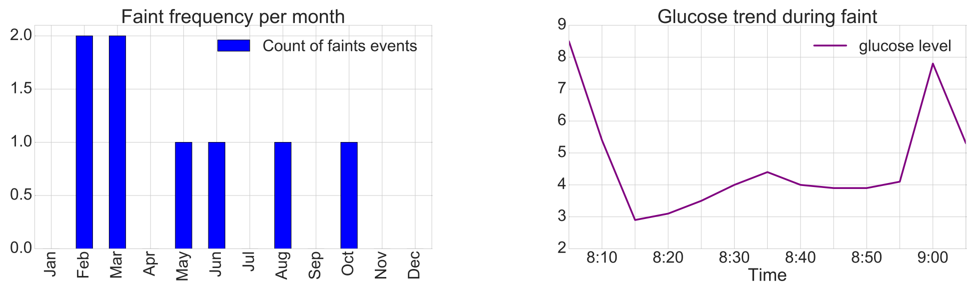
```
% external user query
?- findall(T, (happens_at(faints(P), T), in(T, Month)), Ts).

% internal computation of time with respect to months
in(T, january):-   T > 0,       T =< 75000.
in(T, february):-  T > 75000,   T =< 150000.
in(T, march):-     T > 150000,  T =< 225000.
in(T, april):-     T > 225000,  T =< 300000.
in(T, may):-       T > 300000,  T =< 375000.
in(T, june):-      T > 375000,  T =< 450000.
in(T, july):-      T > 450000,  T =< 525000.
in(T, august):-    T > 525000,  T =< 600000.
in(T, september):- T > 600000,  T =< 675000.
in(T, october):-   T > 675000,  T =< 750000.
in(T, november):-  T > 750000,  T =< 825000.
in(T, december):-  T > 825000,  T =< 900000.
```

LISTING 14. Finding all faint occurrences per month.

Fig. 11(a) depicts the bar graph that is plotted using our activity data.

Glucose trend shows the glucose measurements of the patient during the occurrence of a specific faint. This is useful to identify whether there is a glucose pattern going from bad to



(a) Faint frequency

(b) Glucose trend

FIGURE 11. Supported queries.

worse. Listing 15 shows the queries to extract this information using the knowledge-base. Note that glucose measurements occur every five minutes.

```
?- happens_at(faints(P), T), T > 650000,
   findall(V, (happens_at(measurement(P, glucose, V), Ti),
           T - Ti > 0, T-Ti < 25), Vs).
?- happens_at(faints(P), T), T > 650000,
   findall(V, (happens_at(measurement(P, glucose, V), Ti),
           Ti - T > 0, Ti-T < 40), Vs).
```

LISTING 15. Finding glucose measurements regarding a faint event.

Fig. 11(b) shows the glucose trend of the patient for a specific occurrence of a faint event on Day 299 of the dataset (T = 747521, which corresponds to October), between 8AM and 9AM.

Faint details show various other information related to a faint event. We can identify where the faint happened, and what the patient was doing just before faint happened. We can also answer important questions such as ‘Is there a correlation between the faint event and the time of day, goal or location of the user?’ Listing 16 shows such example queries and results.

```
?- happens_at(faints(P), T), holds_at(location_of(P)=L, T).
P = john,
T = 18,
L = on_street.

?- happens_at(faints(P), T), holds_at(goal(G, P)=active, T).
P = john,
T = 18,
G = at_home.
```

LISTING 16. Details for a faint event.

7. CONCLUSIONS

We have presented an activity recognition capability that is integrated within a logic-based agent architecture to recognise complex activities related to diabetes monitoring. The approach makes use of an *activity lifecycle*, in which activities are treated as temporal fluents that can change state according to events that occur through time. The framework also proposes a *goal lifecycle* for goals that drive activities. The monitoring agent utilises the activity recognition capability to reason about the user's activities given the user's high-level goals. The added value of such monitoring is in the detection of life threatening situtions when the agent advises the user about what to do, it alerts the user's carers about the user's condition and it interacts with the surrounding ambient (when possible) so that the user's location can be easily identified.

We have motivated the work with a specific scenario illustrating how monitoring and recognising activities of a type 1 diabetic patient can be naturally conceived as a computational logic problem. The approach we have developed is particularly suitable for symbolic reasoning agents that use heuristic rules based upon medical guidelines. The main emphasis of the work has been on motivating and conceptually organising the knowledge representation of the recognition in terms of activity and goal lifecycles. In this context, we have evaluated our framework by outlining different ways to carry out the recognition of significant events for a case study with and without these lifecycles. We have shown that our recognition rule regarding a faint event is sound.

Our approach has maintained the split of domain independent versus domain dependent specifications of the original Event Calculus, however, our version is applicable for complex activities as follows. In the original Event Calculus the persistence axioms holds_at/2, holds_for/2 and broken_during/2 are domain independent while the definition of the predicates initiates_at/3, terminates_at/3 and happens_at/2 is domain dependent. In the multi-valued fluents version of the Event Calculus that we use, terminates_at/3 is specified domain independently (as in Artikis et al. (2009)); this optimisation avoids multiple domain dependent definitions of terminates_at/3 for any multi-valued fluents including the activity ones. In addition, in our extension of the multi-valued fluents Event Calculus we have included a domain independent theory about activity and goal lifecycles as specified in Listings 1 & 2, which can be used across domains. Moreover, we have also introduced domain specific causes_at/3 definitions that enabled us to guide the forward reasoning required for caching updates. Our caching mechanism extended the Cached Event Calculus with multi-valued fluents and supports very efficient querying of large event bases.

We have implemented a prototype agent to realize our activity recognition process in GOLEM (Bromuri and Stathis, 2008), the agent platform we used to implement agents in COMMODITY$_{12}$ (Kafalı et al., 2013). Our agent can reason about data produced by our activity generator and raises alerts when life-threatening situations are recognized. However, we have not deployed this agent as a stand alone application for a mobile phone that a diabetic patient can use. Such integration is a crucial next step to perform further experimentation with the proposed system. It will also require to address assumptions that we have made here, viz., that the phone will never be dropped without the user realising it, or that the user is rational and will never ignore the important messages that are precondition to alerts such as calling an ambulance.

We have performed extensive experiments on activities produced by an activity generator that we have developed. We plan to perform further experiments with real activity data, gathering sensory information from the user's mobile phone. Our participation in the COMMODITY$_{12}$ project has given us access to activity data from the clinical trials with actual patients. However, no life threatening situations have arisen during these trials. Therefore, we could not make use of the COMMODITY$_{12}$ data in this work. We have configured our activity generator to produce more than usual hypoglyceamia and faint events to validate the correctness of our framework.

Comparing our approach to supervised models such as HMMs would further highlight our advantages such as expressiveness with the inclusion of temporal constraints and no need for labelled data to construct models. Moreover, exploration of additional domains and datasets would be useful, in particular for smart home environments (Cook and Schmitter-Edgecombe, 2009; van Kasteren et al., 2008).

We have connected the lifecycles of activities and goals in such a way that our framework recognises activities first and then obtains knowledge of goals as part of the context provided by the user. We use the high-level goal of the user to minimise uncertainty created by the user's surroundings. Identifying plans and goals from performed users' activities is also investigated in the literature (Kautz, 1987; Lesh and Etzioni, 1995; Sadri, 2012b). This is particularly significant when agents are performing collaborative activities to achieve a common goal. We believe that a plan recognition approach is helpful and can be incorporated as a complementary model to automatically recognise users' goals rather than asking them to specify those goals. We will further investigate this direction in future work.

We have provided a case study from the healthcare domain, motivated by our participation in the COMMODITY$_{12}$ project. With the addition of domain ontologies, we will investigate how this can be extended and generalised to other domains, where run-time continuous monitoring is essential.

ACKNOWLEDGMENTS

We would like to thank the anonymous referees and Alexander Artikis for their helpful comments on previous versions of this paper. This work has been partially supported by the FP7 project COMMODITY$_{12}$ (www.commodity12.eu), funded by the European Union.

REFERENCES

AGGARWAL, JAKE K., and MICHAEL S. RYOO. 2011. Human activity analysis: A review. ACM Computing Surveys, **43**(3):16:1–16:43.

ALLEN, JAMES F., and GEORGE FERGUSON. 1997. Actions and Events in Interval Temporal Logic, pp. 205–245. Dordrecht: Springer Netherlands.

ARROYO, ROBERTO, J. JAVIER YEBES, LUIS M. BERGASA, IVÁN G. DAZA, and JAVIER ALMAZÁN. 2015. Expert video-surveillance system for real-time detection of suspicious behaviors in shopping malls. Expert Systems with Applications, **42**(21):7991–8005.

ARTIKIS, ALEXANDER, MAREK SERGOT, and GEORGIOS PALIOURAS. 2012. Run-time composite event recognition. *In* Proceedings of the 6th ACM International Conference on Distributed Event-Based Systems (DEBS), ACM, New York, pp. 69–80.

ARTIKIS, ALEXANDER, MAREK SERGOT, and JEREMY PITT. 2009. Specifying norm-governed computational societies. ACM Transactions on Computational Logic, **10**(1):1:1–1:42.

ARTIKIS, ALEXANDER, MAREK J. SERGOT, and GEORGIOS PALIOURAS. 2015. An event calculus for event recognition. IEEE Transactions on Knowledge and Data Engineering, **27**(4):895–908.

AVCI, AKIN, STEPHAN BOSCH, MIHAI MARIN-PERIANU, RALUCA MARIN-PERIANU, and PAUL HAVINGA. 2010. Activity recognition using inertial sensing for healthcare, wellbeing and sports applications: A survey. *In* Proc. of the 23rd International Conf. on Architecture of Computing Systems (ARCS), pp. 1–10.

BANKS, JERRY, JOHN S. CARSON, BARRY L. NELSON, and DAVID M. NICOL. 2010. Discrete-Event System Simulation (5 ed.). Prentice Hall.

BROMURI, STEFANO, and KOSTAS STATHIS. 2008. Situating cognitive agents in GOLEM. *In* Engineering Environment-Mediated Multi-Agent Systems: International Workshop, EEMMAS 2007, Dresden, Germany, October 5, 2007. Selected Revised and Invited Papers. *Edited by* D. Weyns, S. A. Brueckner, and Y. Demazeau. Springer Berlin Heidelberg, pp. 115–134.

BUSS, ARNOLD H. 1996. Modeling with event graphs. *In* Proceedings of the 28th conference on Winter simulation, IEEE Computer Society, pp. 153–160.

CHESANI, FEDERICO, PAOLA MELLO, MARCO MONTALI, and PAOLO TORRONI. 2013. Representing and monitoring social commitments using the event calculus. Autonomous Agents and Multi-Agent Systems, **27**(1):85–130.

CHIAUZZI, EMIL, CARLOS RODARTE, and PRONABESH DASMAHAPATRA. 2015. Patient-centered activity monitoring in the self-management of chronic health conditions. BMC Medicine, **13**(1):1–6.

CHIEU, HAI LEONG, WEE SUN LEE, and LESLIE KAELBLING. 2006. Activity recognition from physiological data using conditional random fields. *In* Singapore-MIT Alliance Symposium.

CHITTARO, LUCA, and ANGELO MONTANARI. 1996. Efficient temporal reasoning in the Cached Event Calculus. Computational Intelligence, **12**:359–382.

CHITTARO, LUCA, MARCO DEL ROSSO, and MICHEL DOJAT. 1995. Modeling medical reasoning with the Event Calculus: An application to the management of mechanical ventilation. *In* Artificial Intelligence in Medicine in Europe (AIME), Volume 934 of *Lecture Notes in Computer Science*, Springer, pp. 79–90.

CLARK, KEITH L. 1987. Negation as failure. *In* Readings in Nonmonotonic Reasoning. Morgan Kaufmann, pp. 311–325. San Francisco.

COOK, DIANE J., and MAUREEN SCHMITTER-EDGECOMBE. 2009. Assessing the quality of activities in a smart environment. Methods of Information in Medicine, **48**(5):480–485.

DESOUZA, CYRUS V., GEREMIA B. BOLLI, and VIVIAN FONSECA. 2010. Hypoglycemia, diabetes, and cardiovascular events. Diabetes Care, **33**(6):1389–1394.

DO, THANG M., SENG W. LOKE, and FEI LIU. 2013. Healthylife: An activity recognition system with smartphone using logic-based stream reasoning. *In* Proceedings of the 9th International Conference on Mobile and Ubiquitous Systems: Computing, Networking, and Services. *Edited by* K. Zheng, M. Li, and H. Jiang. Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 188–199.

DUONG, THI, DINH PHUNG, HUNG BUI, and SVETHA VENKATESH. 2009. Efficient duration and hierarchical modeling for human activity recognition. Artificial Intelligence, **173**(7):830 – 856.

ELHAMOD, MOHANNAD, and MARTIN D. LEVINE. 2013. Automated real-time detection of potentially suspicious behavior in public transport areas. IEEE Transactions on Intelligent Transportation Systems, **14**(2):688–699.

FORTH, JEREMY, KOSTAS STATHIS, and FRANCESCA TONI. 2006. Decision making with a KGP agent system. Journal of Decision Systems, **15**(2-3):241–266.

FRANCOIS, ALEXANDRE R. J., RAM NEVATIA, JERRY R. HOBBS, and ROBERT C. BOLLES. 2005. Verl: An ontology framework for representing and annotating video events. IEEE MultiMedia, **12**(4):76–86.

GEIB, CHRISTOPHER W., VIKAS AGRAWAL, GITA SUKTHANKAR, LOKENDRA SHASTRI, and HUNG HAI BUI. 2015. Architectures for activity recognition and context-aware computing. AI Magazine, **36**(2):3–9.

GORDON, DAWUD, JAN-HENDRIK HANNE, MARTIN BERCHTOLD, ALI ASGHAR NAZARI SHIREHJINI, and MICHAEL BEIGL. 2013. Towards collaborative group activity recognition using mobile devices. Mobile Networks and Applications, **18**(3):326–340.

GU, TAO, ZHANQING WU, LIANG WANG, XIANPING TAO, and JIAN LU. 2009. Mining emerging patterns for recognizing activities of multiple users in pervasive computing. *In* Proceedings of the 6th Annual International Conference on Mobile and Ubiquitous Systems: Computing, Networking and Services (MOBIQUITOUS), pp. 1–10.

HAN, YONGKOO, MANHYUNG HAN, SUNGYOUNG LEE, AM SARKAR, and YOUNG-KOO LEE. 2012. A framework for supervising lifestyle diseases using long-term activity monitoring. Sensors, **12**(5):5363–5379.

HELAL, ABDELSALAM, DIANE J COOK, and MARK SCHMALZ. 2009. Smart home-based health platform for behavioral monitoring and alteration of diabetes patients. Journal of diabetes science and technology, **3**(1):141–148.

HONGENG, SOMBOON, RAM NEVATIA, and FRANCOIS BREMOND. 2004. Video-based event recognition: Activity representation and probabilistic recognition methods. Computer Vision and Image Understanding, **96**(2):129–162.

JAKKULA, VIKRAMADITYA R., and DIANE J. COOK. 2011. Detecting anomalous sensor events in smart home data for enhancing the living experience. *In* Proceedings of the Artificial Intelligence and Smarter Living Workshop, Volume WS-11-07.

JOO, SEONG-WOOK, and RAMA CHELLAPPA. 2006. Recognition of multi-object events using attribute grammars. *In* Proceedings of the International Conference on Image Processing (ICIP), pp. 2897–2900.

KAFALI, ÖZGÜR, STEFANO BROMURI, MICHAL SINDLAR, TOM VAN DER WEIDE, EDUARDO AGUILAR-PELAEZ, ULRICH SCHAECHTLE, BRUNO ALVES, DAMIEN ZUFFEREY, ESTHER RODRGUEZ-VILLEGAS, MICHAEL IGNAZ SCHUMACHER, and KOSTAS STATHIS. 2013. COMMODITY$_{12}$: A smart e-health environment for diabetes management. Journal of Ambient Intelligence and Smart Environments, IOS

Press, **5**(5):479–502.

KAFALI, ÖZGÜR, ALFONSO E. ROMERO, and KOSTAS STATHIS. 2014. Activity recognition for an agent-oriented personal health system. *In* Principles and Practice of Multi-Agent Systems (PRIMA), Volume 8861 of *Lecture Notes in Computer Science*. Springer, pp. 254–269.

KAFALI, ÖZGÜR, ULRICH SCHAECHTLE, and KOSTAS STATHIS. 2014. HYDRA: A hybrid diagnosis and monitoring architecture for diabetes. *In* Proceedings of the 16th IEEE International Conference on E-health Networking, Application & Services (HealthCom), Natal, Brazil.

KAFALI, ÖZGÜR, MICHAL SINDLAR, TOM VAN DER WEIDE, and KOSTAS STATHIS. 2013. $\mathcal{ORC}$: an ontology reasoning component for diabetes. *In* Proceedings of the 2nd International Workshop on Artificial Intelligence and Netmedicine (NetMed).

KAKAS, ANTONIS C., PAOLO MANCARELLA, FARIBA SADRI, KOSTAS STATHIS, and FRANCESCA TONI. 2008. Computational logic foundations of KGP agents. J. Artif. Intell. Res. (JAIR), **33**:285–348.

KAUTZ, HENRY A. 1987. A formal theory of plan recognition. PhD thesis, University of Rochester.

KOURIS, IOANNIS, and DIMITRIS KOUTSOURIS. 2012. A comparative study of pattern recognition classifiers to predict physical activities using smartphones and wearable body sensors. Technology and Health Care, **20**(4):263–275.

KOWALSKI, ROBERT, and MAREK SERGOT. 1986. A logic-based calculus of events. New Generation Computing, **4**(1):67–95.

KRÜGER, FRANK, KRISTINA YORDANOVA, ALBERT HEIN, and THOMAS KIRSTE. 2013. Plan synthesis for probabilistic activity recognition. *In* Proceedings of the 5th International Conference on Agents and Artificial Intelligence (ICAART), pp. 283–288.

LARA, OSCAR D., and MIGUEL A. LABRADOR. 2013. A survey on human activity recognition using wearable sensors. IEEE Communications Surveys & Tutorials, **15**(3):1192–1209.

LESH, NEAL, and OREN ETZIONI. 1995. A sound and fast goal recognizer. *In* Proceedings of the 14th International Joint Conference on Artificial Intelligence (IJCAI), pp. 1704–1710.

LIAO, LIN, DIETER FOX, and HENRY A. KAUTZ. 2007. Extracting places and activities from GPS traces using hierarchical conditional random fields. International Journal of Robotics Research, **26**(1):119–134.

LUŠTREK, MITJA, BOŽIDARA CVETKOVIĆ, VIOLETA MIRCHEVSKA, ÖZGÜR KAFALI, ALFONSO E. ROMERO, and KOSTAS STATHIS. 2015. Recognising lifestyle activities of diabetic patients with a smartphone. *In* Proceedings of the Pervasive Computing Technologies for Healthcare Workshop (PervasiveHealth), pp. 317–324.

MAMDANI, ABE, JEREMY PITT, and KOSTAS STATHIS. 1999. Connected communities from the standpoint of multi-agent systems. New Generation Computing, **17**(4):381–393.

MCCRIMMON, RORY J., and ROBERT S. SHERWIN. 2010. Hypoglycemia in type 1 diabetes. Diabetes, **59**(10):2333–2339.

MINNEN, DAVID, IRFAN ESSA, and THAD STARNER. 2003. Expectation grammars: Leveraging high-level expectations for activity recognition. *In* Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition, Volume 2, pp. 626–632.

MIRSKY, REUTH, YA'AKOV GAL, RONI STERN, and MEIR KALECH. 2016. Sequential plan recognition: Extended abstract. *In* Proceedings of the 2016 International Conference on Autonomous Agents and Multiagent Systems (AAMAS), pp. 1347–1348.

MUKHOPADHYAY, SUBHAS C. 2015. Wearable sensors for human activity monitoring: A review. IEEE Sensors Journal, **15**(3):1321–1330.

NEVATIA, RAM, TAO ZHAO, and SOMBOON HONGENG. 2003. Hierarchical language-based representation of events in video streams. *In* Proceedings of the Computer Vision and Pattern Recognition Workshop (CVPRW), Volume 4, pp. 39–39.

OLIVER, NURIA, ASHUTOSH GARG, and ERIC HORVITZ. 2004. Layered representations for learning and inferring office activity from multiple sensory channels. Computer Vision and Image Understanding, **96**(2):163–180.

PATTERSON, DONALD J., DIETER FOX, HENRY A. KAUTZ, and MATTHAI PHILIPOSE. 2005. Fine-grained activity recognition by aggregating abstract object usage. *In* Proceedings of the Ninth IEEE International Symposium on Wearable Computers (ISWC), pp. 44–51.

RYOO, MICHAEL S, and JAKE K AGGARWAL. 2009. Semantic representation and recognition of continued and recursive human activities. International journal of computer vision, **82**(1):1–24.

SADRI, FARIBA. 2012a. Intention recognition in agents for ambient intelligence: Logic-based approaches.

In Agents and Ambient Intelligence, Volume 12 of *Ambient Intelligence and Smart Environments*. IOS Press, pp. 197–236.

Sadri, Fariba. 2012b. Intention recognition in agents for ambient intelligence: Logic-based approaches. *In* Ambient Intelligence and Smart Environments, Volume 12. IOS Press, pp. 197–236.

Schaffers, Hans, Nicos Komninos, Marc Pallot, Brigitte Trousse, Michael Nilsson, and Alvaro Oliveira. 2011. Smart cities and the future internet: Towards cooperation frameworks for open innovation. *In* The Future Internet, Volume 6656 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, pp. 431–446.

Shoaib, Muhammad, Stephan Bosch, Özlem D. Incel, Hans Scholten, and Paul J. M. Havinga. 2015. A survey of online activity recognition using mobile phones. Sensors, **15**(1):2059.

Stathis, Kostas, Robert Spence, Oscar De Bruijn, and Patrick Purcell. 2006. Ambient intelligence: Agents and interaction in connected communities. *In* The Networked Neighbourhood. *Edited by* P. Purcell. Springer.

Truyen, Tran The, Dinh Q. Phung, Hung Hai Bui, and Svetha Venkatesh. 2008. Hierarchical semi-markov conditional random fields for recursive sequential data. *In* Proceedings of the Twenty-Second Annual Conference on Neural Information Processing Systems, Curran Associates, Inc., pp. 1657–1664.

Turaga, Pavan, Rama Chellappa, Venkatramana S. Subrahmanian, and Octavian Udrea. 2008. Machine recognition of human activities: A survey. IEEE Transactions on Circuits and Systems for Video Technology, **18**(11):1473–1488.

Uzan, Oriel, Reuth Dekel, Or Seri, and Ya'akov Gal. 2015. Plan recognition for exploratory learning environments using interleaved temporal search. AI Magazine, **36**(2):10–21.

van Kasteren, Tim, Athanasios Noulas, Gwenn Englebienne, and Ben Kröse. 2008. Accurate activity recognition in a home setting. *In* Proceedings of the 10th International Conference on Ubiquitous Computing (UbiComp), ACM, New York, NY, USA, pp. 1–9.

van Riemsdijk, M. Birna, Mehdi Dastani, and Michael Winikoff. 2008. Goals in agent systems: A unifying framework. *In* Proceedings of the 7th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS), pp. 713–720.

Viswanath, Bimal, M. Ahmad Bashir, Mark Crovella, Saikat Guha, Krishna P. Gummadi, Balachander Krishnamurthy, and Alan Mislove. 2014. Towards detecting anomalous user behavior in online social networks. *In* Proceedings of the 23rd USENIX Security Symposium, pp. 223–238.

Vu, Van-Thinh, Francois Bremond, and Monique Thonnat. 2003. Automatic video interpretation: A novel algorithm for temporal scenario recognition. *In* Proceedings of the International Joint Conference on Artificial Intelligencee (IJCAI), pp. 1295–1300.

APPENDIX Activity Generation Algorithms

Algorithm 1: Main loop of the generator

Input: initial time: T_0
Result: Event stream of one day of our scenario

$q \leftarrow$ priority_queue() // sorted by time
$q.push(\texttt{at\_home}, T_0)$
$q.push(\texttt{measurement}, T_0)$
while *not* $q.empty()$ **do**
    $current\_event, T \leftarrow q.pop()$
    write_narrative($current\_event$, T)
    $ev, T' \leftarrow$ next_event($current\_event$, T)
    if $ev \neq \emptyset$ **then**
        $q.push(ev, T')$
return

Algorithm 2: procedure write_narrative

```
Input: event: event
Input: time: T
Input: person name: name
Input: glucose value (only for measurements): g
Result: list of Prolog statements: narrative
narrative ← list()
if event == at_home then
    if T == 0 then
        narrative.append(" initially(status_of(name)=ok). ")
        narrative.append(" initially(home_of(name)=h1). ")
        narrative.append(" initially(location_of(h1)=homeAddress). ")
        narrative.append(" initially(location_of(name)=busStop). ")
    narrative.append(" happens_at(adopt_goal_fromGUI(name, at_home(T)), T). ")
else if event == walking or continuing_walk then
    narrative.append(" happens_at(walks(name), T). ")
else if event == chatting_with_friend then
    narrative.append(" happens_at(stands(name), T). ")
else if event == hypo then
    narrative.append(" happens_at(stands(name), T). ")
    narrative.append(" happens_at(raises_alert(name, hypoglyceamia), T). ")
else if event == fainted then
    narrative.append(" happens_at(lies(name), T). ")
else if event == no_response then
    narrative.append(" happens_at(requests(name, confirm_status), T). ")
    narrative.append(" happens_at(no_response(name, 2), T + 2). ")
else if event == recovered then
    narrative.append(" happens_at(requests(name, confirm_status), T). ")
    narrative.append(" happens_at(stands(name), T + 1). ")
    narrative.append(" happens_at(walks(name), T + 2). ")
else if event == arrived then
    narrative.append(" happens_at(arrived(name, homeAddress), T). ")
else if event == measurement then
    narrative.append("happens_at(measurement(name, glucose, g), T). ")
return narrative
```

Algorithm 3: procedure next_event

```
Input: event: current_event
Input: time: T
Input: glucose measurement period: period
Input: probability values: p_chat, p_hypo, p_confirm
Result: (next_event, future time T')
if current_event == at_home then
    return (walking, T+ randint(1, 10))
else if current_event == walking then
    if rand() < p_chat then
        return (chatting_with_friend, T+ randint(1, 10))
    else if rand() < p_hypo then
        return (hypo, T+ random.randint(1, 10))
    else
        return (arrived, T+ randint(15, 25))
else if current_event == chatting_with_friend then
    return (continuing_walk, T+ randint(1, 10))
else if current_event == continuing_walk then
    if rand() < p_hypo then
        return (hypo, T+ random.randint(1, 10))
    else
        return (arrived, T+ randint(5, 15))
else if current_event == hypo then
    if rand() < p_confirm then
        return (recovered, T+ randint(1, 3))
    else
        return (fainted, T + 2)
else if current_event == no_response then
    return (,)                    /* No next event, return empty */
else if current_event == fainted then
    return (arrived, T+ randint(1, 10))
else if current_event == recovered then
    return (arrived, T+ randint(1, 10))
else if current_event == measurement then
    if measurement is the closest to a hypo then
        return (measurement, T + period, glucose=uniform(2.0, 3.7) )
    else
        return (measurement, T + period, glucose=uniform(3.9, 9.0) )
else
                        /* the only remaining case is arrived */
    return (at_home, T + 3000+ randint(1, 100))
```