Activity Recognition for an Agent-oriented Personal Health System

Özgür Kafalı, Alfonso E. Romero and Kostas Stathis

Department of Computer Science
Royal Holloway, University of London
Egham, TW20 0EX, UK
{ozgur.kafali,aeromero,kostas.stathis}@cs.rhul.ac.uk

Abstract. We present a knowledge representation framework that allows an agent situated in an environment to recognise complex activities, reason about their progress and take action to avoid or support their successful completion. Activities are understood as parameterised templates whose parameters consist of a unique name labelling the activity to be recognised, a set of participants co-involved in the carrying out of the activity and a goal revealing the desired outcome the participants seek to bring about. The novelty of the work is the identification of an activity lifecycle where activities are temporal fluents that can be started, interrupted, suspended, resumed, or completed over time. The framework also specifies activity goals and their associated lifecycle, as with activities, and shows how the state of such goals aids the recognition of significant transitions within and between activities. We implement the resulting recognition capability in the Event Calculus and we illustrate how an agent using this capability recognises activities in a personal health system monitoring diabetic patients.

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

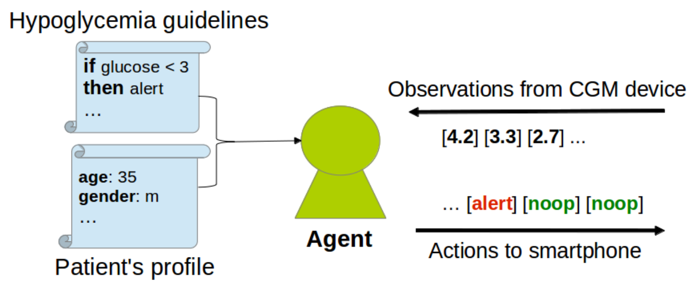


Fig. 1. Continuous Glucose Monitoring (CGM) Agent in COMMODITY$_{12}$ [12].

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 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 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 [15]. Such an agent diagnoses [12], ontologically reasons about [14] and together with specialised agents predict [13] medical emergencies such as *hypoglycemia*. According to the International Classification of Diseases (ICD), hypoglycemia 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 hypoglycemia, 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.

To address conditions such as hypoglycemia we have developed an agent prototype that monitors blood glucose levels of a diabetic patient as shown in Figure 1. The monitoring knowledge and guidelines required for conditions such as hypoglycemia, have been specified using a symbolic, computational logic approach combined with temporal reasoning of the kind supported by the Event Calculus [18]. 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.

The contribution of this work is the specification of an activity recognition capability that is integrated within the logic-based agent architecture discussed in [12] to support reasoning about complex activities from the recognition of basic ones. The capability 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 corroborate hypotheses about conclusions that require further action.

The rest of the paper is structured as follows. We motivate our work in Section 2, by presenting a specific scenario that will provide the rest of the paper a grounding for the ideas presented later. Section 3 reviews the relevant background on activity recognition and the type of approaches followed in a number of applications, not necessarily diabetes. In Section 4, we describe the components that make up our proposal. Section 5 presents the case study and reports performance results. Finally, we conclude the paper with Section 6, where we point out a discussion and possible extensions.

2 The Smart Street Scenario

Consider the following scenario.

> 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's mobile phone app that monitors his diabetes recognised that he has started walking, so it asks John to confirm that he is going back home. After John's confirmation, the app estimates that the walk will be roughly 20-minutes. Halfway, however, John receives an alert informing him that the content of glucose in his blood is abnormally low (a hypoglycemia medical emergency). John did not have enough time to respond to this alert as he passed out and fell on the pavement. Immediately after John falling on the pavement, his doctor and family were informed, an ambulance was called and the nearest street light started flashing to attract attention of passers-by and help the ambulance locate John.

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

The scenario above requires that when the glucose level was low the agent has taken a number of important steps. Immediately after sending the hypoglycemia alert, the agent also sent a message on the app's display asking whether John was feeling ok. As John did not respond to this message because he fainted. This was recognised by the agent because the person had fallen while he was suffering a hypoglycemia attack. As a result, the agent alerted first John's doctor, then John's family and an ambulance giving John's location. The scenario assuming further a neighbourhood e-infrastructure of the kind envisaged in Smart Cities [24]. 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.

3 Activity Recognition

Activity recognition can be defined, in broad terms, as the task of automatically detecting different human behaviours from certain given input data. In the last years, many computational approaches have been proposed to solve this problem (see [25, 4, 1] and references therein). From the point of view of the type of input data received, an activity recognition usually belongs to one of these two main groups: video-based [1, 25], where a computer is used to recognise a sequence of images with one or more persons 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 (mobile telephone, activity tracker, medical sensor, etc.) In this work we will mainly focus on the latter.

It can be noted that the definition of activity recognition given above is essentially too abstract and not many attempts to formalise this problem can be found. In this manner, the task of Activity Recognition is even treated as a "subproblem of computer vision" or of the field where it is applied, and it is not treated in itself. Apart of the lack of formal definitions of the task itself, activities are taken as primitive concepts, not dealing the majority of the available references with a proper definition of them, and just focusing on the computational solution of the problem. The definition of activity, in the general sense, remains an open question which we try to partially address in this work.

Despite this lack of formalisation, links with activity theory have been established mostly within the field of human computer interaction [16]. On the other hand, the contribution of Chen *et al.* [5] presents a formal framework of activities, which are modelled as concepts of an ontology, having specific relations among them, and which are later used to include semantic information into the model of activity recognition proposed.

From an operational perspective, most of the current approaches to activity recognition work follow a hierarchical scheme. This scheme is summarised in Figure 2: first, a stream of data coming from mobile sensors and other sources is available (in our previous example, Jonh's smartphone and the CGM). Second, this raw data is 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* (also known as *actions* in the literature [25]). These events for our example would correspond to simple physical actions such as *walk*, *stand*, *lie*. Finally, the primitive events together with the context (such as historical information and user conformations) are used to recognise (more complex) activities, represented in terms of the events which

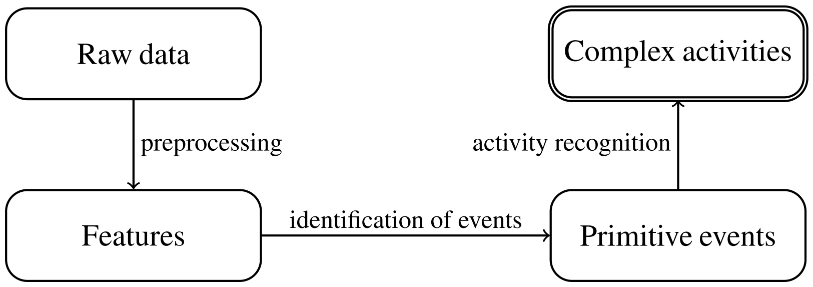


Fig. 2. General data flow for activity recognition.

were captured in the previous level. For our concrete case, these are activities which are suitable for being monitored or treated by doctors (like for instance, the case that John has fainted). In this work we will mainly focus on the last step, represented in Figure 2 with a double box.

The methods for preprocessing raw data are highly dependent on the device type and its parameters, and we will not deal with them here. In order to detect basic events many alternatives techniques have been used: mainly supervised learning methods over tagged data (such as Hidden Markov Models, decision trees, neural networks). This subproblem has been successfully address by the previously mention techniques, resulting in very high values of accuracy (an exhaustive list is collected in [1] showing performances from 85% to 95%).

Next, we will review more carefully the methods for complex activity recognition, fundamentally those dealing with symbolic approaches. Apart from statistical models, two main solutions for this task has been proposed: syntactic methods and description-based methods. In syntactic methods, activities are defined as production rules of a grammar, reducing the problem of recognition to the one of parsing. In order to attain for uncertainty methods such as stochastic context grammars have often been used [20]. Joo and Chellappa propose a framework for recognition of events using attribute grammars [11]. They represent sequence of events as grammar rules as well as assigning attributes to each event. 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 telling 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. While their framework can successfully recognise such activities, it's not equipped to deal with the types of scenarios we have discussed in this paper, where the duration of an event and other contextual information are significant to recognition.

Ontologies are also utilised to represent and recognise events. In [8], the authors present an ontology based on the video event representation language (VERL). They use a logic-based markup language to represent composite events as sequences of primitive events as well using interval logic to capture temporal relations among events. They present a scenario where people are recognised while tailgating through a door. However, their rules are not as representative as ours and do not take into account contextual information. Nevatia *et al.* [21] define an ontology and a language to annotate video and describe complex activities as a function of simple activities. A very similar approach is taken in [26] where a symbolic approach to recognise temporal scenarios by specifying all its elements (e.g., characters, sub-scenarios, constraints) is presented.

Artikis *et al.* [3] study a variant of the Event Calculus for recognising composite events at run-rime using a stream of time-stamped simple derived events. In this system recognition of higher-level activities are treated as recognition of composite events but an activity (composite event) lifecycle as the one specified in section 4.2 is missing. So we can think of our framework as being more methodological for a specific class of applications where the goal achieved by an activity is an important requirement for the background knowledge of the recognition process. Knowing in advance the goals of participants in activities is an important consideration for applications such as the one we consider here, as they provide important contextual information in support of recognition. However, we do not recognise goals (or more generally the intentions of participants, as in [23]). Instead, we monitor an activity given the activity's goal and, where possible, we check that there are no activities or events that may interfere with the achievement of that goal.

There is also an important number of publications dealing with activity recognition and healthcare. For instance, a comparative of machine learning predictors for low-level activity recognition using body sensors is presented in [17]. In this study a lot of effort is made on the recognition of low-level events (which is done using different machine learning methods such as decision trees or Bayesian networks) and little is shown about the possible extension of recognising higher-level (complex) activities. Given the inherent risk of the application, most of the literature in this sub-field (and, in general, in sensor-based human activity recognition [19]) actually deals with recognition of low-level events, trying to find models achieving the minimum error.

For the specific case of diabetes not much work has been reported in the literature. One related work, where a system for monitoring diabetic patients (with an activity-recognition module) is presented in [10]. While there is an important description of the architecture of the system (e.g., the context of a smart home), there is not much discussion about the list of possible activities that could be recognised for this case of a diabetic patient, or about the different alternatives for activity recognition. The authors base their approach on Hidden Markov Models. Another interesting contribution is done by Han *et al.* [9], where the concept of "Disease Influenced Activity" is presented. Like many others, this contribution is also focused on monitoring uncommon patters (e.g., "frequent drinking" for diabetes) and presenting them to the doctor. In their approach they also make use of a Machine learning algorithm.

4 The Activity Recognition Framework

4.1 Architecture

Figure 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

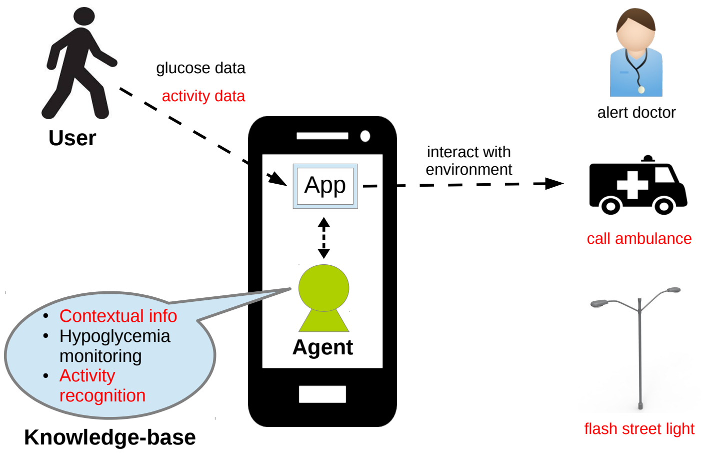


Fig. 3. Diabetes Monitoring and Management in our system. Components shown in red are the extended features for the Continuous Monitoring Agent.

system (as shown in Figure 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) as well as contextual information about the user's environment (e.g., the user's current goal). The application also allows the agent to interact with the user's surroundings. In case of an emergency, the agent can call an ambulance and flash the street lights to attract attention as well as alerting the user's doctor.

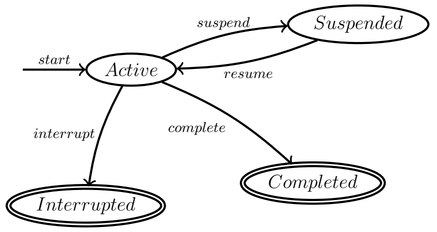
4.2 Recognising activities and their lifecycle transitions in the Event Calculus

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 treats activities as temporal fluents that can be started, interrupted, suspended, resumed, or completed in time. The framework also proposes a template for activity goals and their associated lifecycle, similar to that of activities. Both lifecycles are presented in Fig.4.

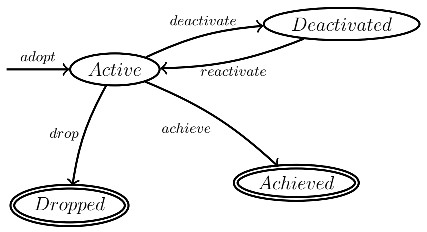
We assume the notion of primitive events (e.g., walks, stands, lies), which are represented as input from the low level recognition system (see Fig.2). The framework differentiates between events, activities and activity transitions, caused by special events

Predicate	Description
$happens\_at(E, T)$	Event E happens at time T
$initially(F = V)$	Fluent F has value V at time 0
$holds\_at(F = V, T)$	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

Table 1. Domain-independent axioms of the Event Calculus.



(a) Activity lifecycle



(b) Goal lifecycle

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

within an activity and according to the activity's lifecycle, or changes between activities. For example, the primitive event that a person stands provided as an input observation from a sensor, terminates the status of the activity of walking from active to suspended, and initiates standing.

To interpret and reason about events and activities we use the Event Calculus [18]. Table 1 summarises the domain-independent axioms used of the Event Calculus; these axioms assume multi-valued fluents as discussed in [2]. On top of the domain-independent axioms, our framework consists of the following additional components:

– an activity theory that regulates the activity lifecycle;
– a goal theory that regulates 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 for the domain model and the event narrative). Figure 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 [22] 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*.

Listing 1 presents the Event Calculus axioms specifying the domain independent activity theory. Lines (1-5) describe the events that happen when an activity is recognised to have been started (`started_at/2`), suspended (`suspended_at/2`), resumed (`resumed_at/2`), interrupted (`interrupted_at/2`), or eventually been completed (`completed_at/2`) at a specific time. Lines (7-11) 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 [2] (axiom 19).

```
1  happens_at(start(Activity), T):- started_at(Activity, T).
2  happens_at(suspend(Activity), T):- suspended_at(Activity, T).
3  happens_at(resume(Activity, T):- resumed_at(Activity, T).
4  happens_at(interrupt(Activity, T):- interrupted_at(Activity, T).
5  happens_at(complete(Activity), T):- completed_at(Activity, T).

7  initiates_at(start(A), A=active, T).
8  initiates_at(suspend(A), A=suspended, T).
9  initiates_at(resume(A), A=active, T).
10 initiates_at(interrupt(A), A=interrupted, T).
11 initiates_at(complete(A), A=completed, T).
```

Listing 1. Domain independent activity theory in RETRACT.

Similar to the activity theory, Listing 2 presents the Event Calculus axioms for the goal theory. Lines (1-5) describe the events that happen when a goal is said to have been adopted (`adopted_at/2`), deactivated (`deactivated_at/2`), reactivated (`reactivated_at/2`), dropped (`dropped_at/2`), or eventually been achieved (`achieved_at/2`) at a specific time. Lines (7-11) describe now describe how the goal events initiate different values for the goal fluents.

```
1  happens_at(adopt(Goal), T):- adopted_at(Goal, T).
2  happens_at(deactivate(Goal), T):- deactivated_at(Goal, T).
3  happens_at(reactivate(Goal, T):- reactivated_at(Goal, T).
4  happens_at(drop(Goal, T):- dropped_at(Goal, T).
5  happens_at(achieve(Goal), T):- achieved_at(Goal, T).

7  initiates_at(adopt(G), G=active, T).
8  initiates_at(deactivate(G), G=deactivated, T).
9  initiates_at(reactivate(G), G=active, T).
10 initiates_at(drop(G), G=dropped, T).
11 initiates_at(achieve(G), G=achieved, T).
```

Listing 2. Domain independent goal theory in RETRACT.

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 value of the fluent, drawn from the set of possible values $active$, $suspended$, $interrupted$ and $completed$. We represent similarly a goal fluent as $goal(Name, Participants) = State$. The $Name$ (e.g. at_home) and the $Participants$ (e.g. $[john, peter]$) are defined like those of the activity fluent, what changes now is the current value of the $State$, drawn from the set of possible values $active$, $deactivated$, $dropped$ and $achieved$.

```
1  started_at(activity(walking, P, G), T):-
2      holds_at(goal(G, P)=active, T),
3      happens_at(walks(P), T).

5  suspended_at(activity(walking, P, G)), T):-
6      happens_at(stands(P), T),
7      holds_at(activity(walking, P, G)=active, T).

9  resumed_at(activity(walking, P, G), T):-
10     holds_at(activity(walking, P, G)=suspended, T),
11     happens_at(walks(P), T).
12 ...

14 interrupted_at(activity(A, P, G)), T):- happens_at(drop(goal(G, P)), T).

16 completed_at(activity(A, P, G)), T):- happens_at(achieved(goal(G, P)), T).
```

Listing 3. An example of domain dependent activity theory.

```
1  adopted_at(goal(G, P), T):- happens_at(adopt_goal_fromGUI(P, G), T).

3  deactivated_at(goal(G, P), T):- happens_at(deactivate_goal_fromGUI(P, G), T).

5  reactivated_at(goal(G, P), T):- happens_at(reactivate_goal_fromGUI(P, G), T).

7  dropped_at(goal(G, P), T):- happens_at(drop_goal_fromGUI(P, G), T).

9  achieved_at(goal(at_home, P), T):-
10     holds_at(location_of(P, L)=true, T),
11     holds_at(home_of(P, H)=true, T),
12     holds_at(location_of(H, L)=true, T).
```

Listing 4. An example of domain dependent goal theory.

Listing 3 shows an extract of the domain dependent activity theory exemplified, in part, by the activity of $walking$. This is started once a low-level event $walks(P)$ happens (stating that the participant P walks, see lines 1-3). We assume that the low-level activity recognition module will not send us more low-level $walk(P)$ 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 suspend the current activity. Lines (5-7) show how standing suspends walking. The $walking$ activity is resumed (becomes *active* again) when a low-level $walks(P)$ event happens (Lines 9-11). Any activity is interrupted when that activity's goal is dropped (Line 14), and, any activity is *completed* when that activity's goal has been achieved (Line 16).

Listing 4 shows an extract of a domain dependent goal theory exemplified, in part, by the goal of at_home. In this domain, we assume that the user manages directly the

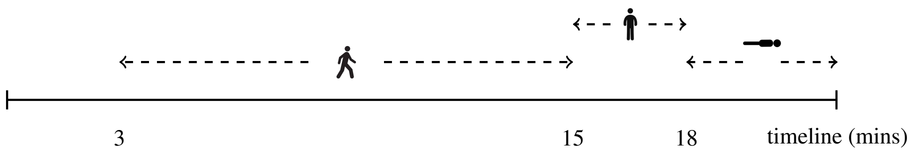


Fig. 5. John's activities.

goal from the graphical user interface (GUI) of the application. For any goal, actions of the user at the GUI are interpreted as internal events that cause the adoption of a new goal (Line 1) or the deactivation/reactivation/dropping of an existing goal (Lines 3, 5, and 7 respectively). Only the achievement of a goal is specified case by case; Line 9-12 shows an example of when the at_home goal is achieved.

5 Case Study

We now focus on the scenario described in Section 2. Let us first see the primitive events that lead to John falling on the street due to a hypoglycemia episode. Figure 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 (see Table 1). It represents the validity period for activities that are in active or suspended state. This is shown in Listing 5.

```
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, [3,16]).
holds_for(activity(standing, john, null)=active, [16,19]).
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 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):-
    holds_for(activity(walking, Person, _)=active, [_,T1]),
    holds_for(activity(standing, Person, _)=active, [T2,T3]),
    holds_for(activity(lying, Person, _)=active, [T,_]),
    immediately_before(T1,T2),
    immediately_before(T3,T).

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

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

Listing 7 improves the previous rule with the use of the activity theory. 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 activities as previously done with the $immediately\_before/2$ predicate shown in Listing 6.

```
happens_at(falls(Person), T):-
    holds_for(activity(lying, Person, _)=active, [T,_]),
    holds_for(activity(standing, Person, _)=suspended, [T,_]),
    holds_at(activity(walking, Person, _)=suspended, T).
```

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

In order to recognise that John has fainted, we must have 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 hypoglycemia alert and stops to look at his smartphone. Unfortunately, he fell down soon after checking the alert. The agent running on his smartphone asks for John's status immediately after he fell.

```
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. Listing 9 describes these rules. We capture fainting as a special case of the fall event (e.g., the interruption of walking). 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:

– John has the goal of walking and it has not been achieved yet;
– the agent has sent an alert to John following a hypoglycemia before he fell;
– the agent has asked John to confirm his status soon after he fell, and it has not received a response.

```
happens_at(faints(Person), T):-
    happens_at(raises_alert(Person, hypoglycemia), T1),
    happens_at(falls(Person), T2),
    happens_at(requests(Person, confirm_status), T),
    \+ happens_at(response(Person, status_ok), _),
    holds_at(goal(at_home, Person)=active, T),
    T1 < T2,
    T2 < T.
```

Listing 9. Recognition of a faint event.

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 connects 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 assisting during a faint event.

Implementation and supported queries

We have implemented a prototype of the framework for the domain described in Section 5. We have used tuProlog for the implementation of the Event Calculus and Java to read the datasets generated for testing purposes.

```
Query 1
happens_at(falls(Person), T). --> yes.
Person / john  T / 19

Query 2
happens_at(falls(john), 15). --> no.

Query 3
happens_at(faints(john), T). --> yes.
T / 20

Query 4
happens_at(faints(Person), 20). --> yes.
Person / john
```

Listing 11. Supported queries.

Listing 11 reports the different types of queries for *falls* and *faints* events. We have evaluated the framework with both grounded queries and queries involving variables. Our implementation of the Event Calculus allows fast query times and is able to answer queries with time given as a variable, e.g., *happens_at(faints(john), T)*.

6 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 activity goals, similar to that of activities. The activity recognition capability then supports the monitoring agent to reason upon the link between the patient's activities and goals using their corresponding lifecycles, and provides advice and explanation to the patient, as well as detecting emergency situations.

We have motivated the work with a specific scenario illustrating how monitoring and recognising activities of a diabetic patient can be naturally conceived as a multi-agent systems problem. The approach we have developed is particularly suitable for symbolic reasoning agents that use monitoring observations according to medical guidelines, even in the light of incomplete information, and can take into account information that refers to the lifestyle of the patient.

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 proposed framework by outlining different ways to carry out the recognition of significant events for a case study with and without these lifecycles. We have also compared our work in the context of existing activity recognition frameworks and we have discussed how the key aspects of our framework extend the most relevant existing work.

As we have concentrated on the knowledge representation of complex activities we have decided not to carry out any performance evaluation of our Event Calculus implementation. The main reason for this choice is that Event Calculus performance is not an obstacle in the development of practical applications, since we could have used an *off-the-shelf* approach, for example see [3, 6, 7]. However, we believe that our version of the Event Calculus has merits, especially if combined with our recognition and agent monitoring framework, but this discussion is beyond the limited space of this paper. As part of future work, we plan to compare the performance of our implementation of the Event Calculus with similar approaches such as RTEC [3] and $\mathcal{REC}$ [6].

We have connected the lifecycle of an activity and a goal using one direction only, viz., our framework recognises activities first and then obtains knowledge of goals as part of the context provided by the patient (user). The other direction is also interesting, e.g., recognise goals from performed activities as described in [23]. This is particularly significant when agents are performing collaborative activities to achieve a common goal. Here, we have presented a simple goal structure. We will investigate how this can be extended and generalised with the integration of domain ontologies showing how our approach can be extended to other domains where run-time continuous monitoring is essential.

Acknowledgements

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

References

1. Aggarwal, J., Ryoo, M.: Human activity analysis: A review. ACM Comput. Surv. 43(3), 16:1–16:43 (2011)
2. Artikis, A., Sergot, M., Pitt, J.: Specifying norm-governed computational societies. ACM Trans. Comput. Logic 10(1), 1:1–1:42 (Jan 2009)
3. Artikis, A., Sergot, M., Paliouras, G.: Run-time composite event recognition. In: Proceedings of the 6th ACM International Conference on Distributed Event-Based Systems. pp. 69–80. DEBS '12, ACM, New York, NY, USA (2012)
4. Avci, A., Bosch, S., Marin-Perianu, M., Marin-Perianu, R., Havinga, P.: Activity recognition using inertial sensing for healthcare, wellbeing and sports applications: A survey. In: Architecture of computing systems (ARCS), 2010 23rd international conference on. pp. 1–10. VDE (2010)
5. Chen, L., Nugent, C.D., Wang, H.: A knowledge-driven approach to activity recognition in smart homes. Knowledge and Data Engineering, IEEE Transactions on 24(6), 961–974 (2012)
6. Chesani, F., Mello, P., Montali, M., Torroni, P.: A Logic-Based, Reactive Calculus of Events. In: Gavanelli, M., Riguzzi, F. (eds.) 24th Convegno Italiano di Logica Computazionale (CILC09) (2009)
7. Chittaro, L., Montanari1, A.: Efficient temporal reasoning in the cached event calculus. Computational Intelligence 12(3), 359–382 (1996)
8. Francois, A.R.J., Nevatia, R., Hobbs, J.R., Bolles, R.C.: Verl: An ontology framework for representing and annotating video events. IEEE MultiMedia 12(4), 76–86 (2005)
9. Han, Y., Han, M., Lee, S., Sarkar, A., Lee, Y.K.: A framework for supervising lifestyle diseases using long-term activity monitoring. Sensors 12(5), 5363–5379 (2012)
10. Helal, A., Cook, D.J., Schmalz, M.: Smart home-based health platform for behavioral monitoring and alteration of diabetes patients. Journal of diabetes science and technology 3(1), 141–148 (2009)
11. Joo, S.W., Chellappa, R.: Recognition of multi-object events using attribute grammars. In: ICIP. pp. 2897–2900 (2006)
12. Kafalı, Ö., Bromuri, S., Sindlar, M., van der Weide, T., Aguilar-Pelaez, E., Schaechtle, U., Alves, B., Zufferey, D., Rodrguez-Villegas, E., Schumacher, M.I., Stathis, K.: COMMODITY$_{12}$: A smart e-health environment for diabetes management. Journal of Ambient Intelligence and Smart Environments, IOS Press 5(5), 479–502 (2013)
13. Kafalı, Ö., Schaechtle, U., Stathis, K.: HYDRA: a HYbrid Diagnosis and monitoRing Architecture for diabetes. In: 16th International Conference on E-health Networking, Application & Services (HealthCom'14). IEEE, Natal, RN, Brazil (15-18 Oct 2014)
14. Kafalı, Ö., Sindlar, M., van der Weide, T., Stathis, K.: $\mathcal{ORC}$: an $\mathcal{O}$ntology $\mathcal{R}$easoning $\mathcal{C}$omponent for diabetes. In: 2nd International Workshop on Artificial Intelligence and Netmedicine (NetMed'13) (2013)
15. Kakas, A.C., Mancarella, P., Sadri, F., Stathis, K., Toni, F.: Computational logic foundations of KGP agents. J. Artif. Intell. Res. (JAIR) 33, 285–348 (2008)
16. Kaptelinin, V.: Activity theory: Implications for human-computer interaction. Context and consciousness: Activity theory and human-computer interaction pp. 103–116 (1996)
17. Kouris, I., Koutsouris, D.: 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 (2012)
18. Kowalski, R., Sergot, M.: A logic-based calculus of events. New Generation Computing 4(1), 67–95 (1986)

19. Lara, O.D., Labrador, M.A.: A survey on human activity recognition using wearable sensors. Communications Surveys & Tutorials, IEEE 15(3), 1192–1209 (2013)
20. Minnen, D., Essa, I., Starner, T.: Expectation grammars: Leveraging high-level expectations for activity recognition. In: Computer Vision and Pattern Recognition, 2003. Proceedings. 2003 IEEE Computer Society Conference on. vol. 2, pp. II–626. IEEE (2003)
21. Nevatia, R., Zhao, T., Hongeng, S.: Hierarchical language-based representation of events in video streams. In: Computer Vision and Pattern Recognition Workshop, 2003. CVPRW'03. Conference on. vol. 4, pp. 39–39. IEEE (2003)
22. van Riemsdijk, M.B., Dastani, M., Winikoff, M.: Goals in agent systems: A unifying framework. In: Proceedings of the 7th International Joint Conference on Autonomous Agents and Multiagent Systems - Volume 2. pp. 713–720. AAMAS '08 (2008)
23. Sadri, F.: Intention recognition in agents for ambient intelligence: Logic-based approaches. In: Bosse, T. (ed.) Agents and Ambient Intelligence, Ambient Intelligence and Smart Environments, vol. 12, pp. 197–236. IOS Press (2012)
24. Schaffers, H., Komninos, N., Pallot, M., Trousse, B., Nilsson, M., Oliveira, A.: Smart cities and the future internet: Towards cooperation frameworks for open innovation. In: Future Internet Assembly. pp. 431–446 (2011)
25. Turaga, P., Chellappa, R., Subrahmanian, V.S., Udrea, O.: Machine recognition of human activities: A survey. Circuits and Systems for Video Technology, IEEE Transactions on 18(11), 1473–1488 (2008)
26. Vu, V.T., Bremond, F., Thonnat, M.: Automatic video interpretation: A novel algorithm for temporal scenario recognition. In: IJCAI. vol. 3, pp. 1295–1300 (2003)