Towards A Simple Event Calculus for Run-Time Reasoning

Christos Vlassopoulos[1,2] and **Alexander Artikis**[3,1]
[1]Institute of Informatics and Telecommunications, NCSR "Demokritos", Athens, Greece
[2]Department of Informatics and Telecommunications, National and Kapodistrian University of Athens, Greece
[3]Department of Maritime Studies, University of Piraeus, Greece

Abstract

The Event Calculus for Run-Time reasoning (RTEC) is a logic programming implementation of the Event Calculus, designed to compute continuous narrative assimilation queries on data streams. RTEC has been used for complex event recognition in various applications domains, such as maritime monitoring, city transport management and human activity recognition. The construction of the complex event definitions has proven a time-consuming process, since it requires several interactions with domain experts (e.g. transport engineers). To address this issue, we present a simple language for RTEC, with the aim of supporting people who are not familiar with the Event Calculus or (logic) programming. A compiler translates, in a process transparent to the user, a specification in the simple language to an RTEC event description that may be subsequently used for continuous query computation.

Introduction

Today's organisations need to act upon high-velocity data streams in order to capitalise on opportunities and detect threats. Towards this, event recognition systems have been particularly helpful, as they support the detection of 'complex events' (CE) of special significance, given streams of 'simple, derived events' (SDE) arriving from various types of sensor (Artikis et al. 2012). The 'definition' of a CE imposes temporal and, possibly, atemporal constraints on its subevents, i.e. SDEs or other CEs. In the maritime domain, for example, CE recognition systems have been used to make sense of position streams emitted from thousands of vessels, in order to detect, in real-time, suspicious and illegal activity that may have dire effects in the maritime ecosystem and passenger safety.

Several CE recognition frameworks have been proposed, supporting data streams by means of various optimisation techniques (Cugola and Margara 2012; Bicocchi et al. 2014). One such framework is the Event Calculus for Run-Time reasoning (RTEC), a logic programming implementation of the Event Calculus (Kowalski and Sergot 1986), designed to compute continuous narrative assimilation queries on data streams (Artikis, Sergot, and Paliouras 2015). RTEC includes novel implementation techniques for efficient CE recognition. A form of caching stores the results of subcomputations in the computer memory to avoid unnecessary re-computations. A set of interval manipulation constructs simplify CE definitions and improve reasoning efficiency. An indexing mechanism makes RTEC robust to SDEs that are irrelevant to the CEs we want to recognise and so RTEC can operate without SDE filtering modules. Finally, a 'windowing' mechanism supports real-time CE recognition.

RTEC has been used for CE recognition in various applications domains, including human activity recognition from video content, city transport management, and maritime monitoring (Patroumpas et al. 2017). The construction of CE definitions was performed manually, since annotated data were insufficient for employing machine learning techniques. This was a time-consuming process that required several interactions with domain experts. To address this issue, we have been developing a simpler language for RTEC, with the aim of supporting people who are not familiar with the Event Calculus or (logic) programming. A compiler translates, in a process transparent to the user, a specification in the simpler language to an RTEC event description that may be subsequently used for query computation.

The remainder of the paper is structured as follows. First, we present RTEC. Then, we discuss the simple language and illustrate its use through various examples. In the section that follows, we describe the functionality of the compiler that translates the statements of the simple language to RTEC. Subsequently, we present the empirical evaluation of our approach, and briefly discuss related and further work.

RTEC

The Event Calculus for Run-Time reasoning (RTEC) is a logic programming implementation of the Event Calculus, designed to compute continuous narrative assimilation queries on data streams for complex event (CE) recognition. The time model is linear and includes integer time-points. Following Prolog, variables start with an upper-case letter, while predicates and constants start with a lower-case letter. Where F is a *fluent*—a property that is allowed to have different values at different points in time—the term $F = V$ denotes that fluent F has value V. Boolean fluents are a special case in which the possible values are true and false. Informally, $F = V$ holds at a particular time-point if $F = V$ has been initiated by an event at some earlier time-point, and not terminated by another event in the meantime.

An *event description* in RTEC includes rules that define

Predicate	Meaning
happensAt(E, T)	Event E occurs at time T
holdsAt$(F = V, T)$	The value of fluent F is V at time T
holdsFor$(F = V, I)$	I is the list of the maximal intervals for which $F = V$ holds continuously
initiatedAt$(F = V, T)$	At time T a period of time for which $F = V$ is initiated
terminatedAt$(F = V, T)$	At time T a period of time for which $F = V$ is terminated
union_all(L, I)	I is the list of maximal intervals produced by the union of the lists of maximal intervals of list L
intersect_all(L, I)	I is the list of maximal intervals produced by the intersection of the lists of maximal intervals of list L
relative_complement_all_(I', L, I)	I is the list of maximal intervals produced by the relative complement of the list of maximal intervals I' with respect to every list of maximal intervals of list L

Table 1: Main predicates of RTEC.

the event instances with the use of the happensAt predicate, the effects of events with the use of the initiatedAt and terminatedAt predicates, and the values of the fluents with the use of the holdsAt and holdsFor predicates, as well as other, possibly atemporal, constraints. Table 1 summarises the main predicates of RTEC. The code is available at `https://github.com/aartikis/RTEC`. We represent instantaneous SDEs and CEs by means of happensAt, while durative CEs are represented as fluents. The majority of CEs are durative and, therefore, in CE recognition the task generally is to compute the maximal intervals for which a fluent expressing a CE has a particular value continuously. Below, we discuss the representation of fluents, and present the way we compute their maximal intervals.

Fluents

holdsFor$(F = V,\ I)$ represents that I is the list of the maximal intervals for which fluent F has value V continuously. holdsAt$(F = V,\ T)$ represents that fluent F has value V at a particular time-point T. holdsAt and holdsFor are defined in such a way that, for any fluent F, holdsAt$(F = V,\ T)$ if and only if time-point T belongs to one of the maximal intervals of I for which holdsFor$(F = V,\ I)$. Fluents in RTEC are of two kinds: *simple* and *statically determined*. We assume, without loss of generality, that these types are disjoint.

Simple fluents For a simple fluent F, $F = V$ holds at a time-point T if $F = V$ has been initiated by an event at some time-point earlier than T, and has not been terminated at some other time-point in the meantime. This is an implementation of the law of inertia. The time-points at which $F = V$ is initiated are computed with the use of initiatedAt rules, which have the following form (terminatedAt are defined similarly):

$$\begin{aligned}&\text{initiatedAt}(F = V,\ T) :-\\ &\quad \text{happensAt}(E,\ T),\\ &\quad conditions[T]\end{aligned}$$

The $conditions[T]$ set includes further constraints on time-point T, expressed as follows:

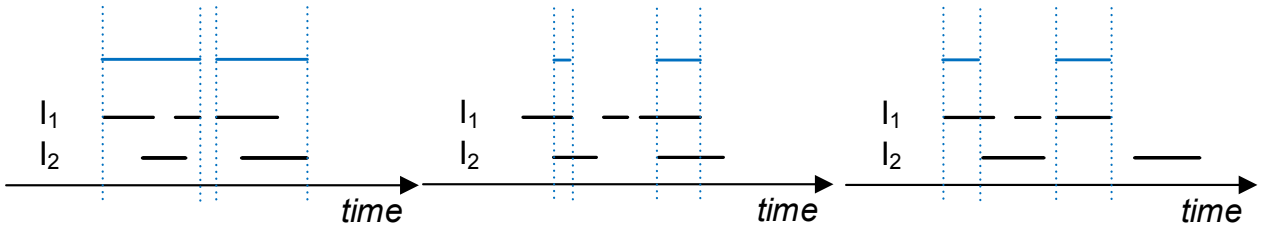
- a possibly empty set of happensAt predicates expressing constraints on the (non-)occurrence of events;
- a possibly empty set of holdsAt predicates expressing constraints on fluents; and
- a possibly empty set of atemporal constraints.

Consider the following example from human activity recognition on video content:

$$\begin{aligned}&\text{initiatedAt}(moving(P_1, P_2) = \text{true},\ T) :-\\ &\quad \text{happensAt}(\text{start}(walking(P_1) = \text{true}),\ T),\\ &\quad \text{holdsAt}(walking(P_2) = \text{true},\ T),\\ &\quad \text{holdsAt}(close(P_1, P_2) = \text{true},\ T).\\ &\text{initiatedAt}(moving(P_1, P_2) = \text{true},\ T) :-\\ &\quad \text{happensAt}(\text{start}(walking(P_2) = \text{true}),\ T),\\ &\quad \text{holdsAt}(walking(P_1) = \text{true},\ T),\\ &\quad \text{holdsAt}(close(P_1, P_2) = \text{true},\ T).\\ &\text{initiatedAt}(moving(P_1, P_2) = \text{true},\ T) :-\\ &\quad \text{happensAt}(\text{start}(close(P_1, P_2) = \text{true}),\ T),\\ &\quad \text{holdsAt}(walking(P_1) = \text{true},\ T),\\ &\quad \text{holdsAt}(walking(P_2) = \text{true},\ T).\\ &\text{terminatedAt}(moving(P_1, P_2) = \text{true},\ T) :-\\ &\quad \text{happensAt}(\text{end}(walking(P_1) = \text{true}),\ T).\\ &\text{terminatedAt}(moving(P_1, P_2) = \text{true},\ T) :-\\ &\quad \text{happensAt}(\text{end}(walking(P_2) = \text{true}),\ T).\\ &\text{terminatedAt}(moving(P_1, P_2) = \text{true},\ T) :-\\ &\quad \text{happensAt}(\text{end}(close(P_1, P_2) = \text{true}),\ T).\end{aligned} \tag{1}$$

$walking$ is a durative SDE detected on video frames. (Recall that SDEs are given as input to RTEC.) start$(F = V)$ (resp. end$(F = V)$) is a built-in RTEC event taking place at each starting (ending) point of each maximal interval for which $F = V$ holds continuously. $close(A, B) = \text{true}$ when the distance between tracked entities (people and/or objects) A and B does not exceed some threshold of pixel positions. The above formalisation states that P_1 is moving with P_2 when they are walking close to each other.

Note that in this formulation of the Event Calculus, terminatedAt$(F = V,\ T)$ does not necessarily imply that $F = V$ at T. Similarly, initiatedAt$(F = V,\ T)$ does not necessarily imply that $F \neq V$ at T.



(a) Union. (b) Intersection. (c) Relative Complement.

Figure 1: A visual illustration of the three interval manipulation constructs of RTEC. In this example, there are two input fluent streams, I_1 and I_2. The output of each interval manipulation construct is colored light blue.

To compute $\mathsf{holdsFor}(moving(P_1, P_2) = \mathsf{true}, I)$, that is, to compute the maximal intervals for which $moving(P_1, P_2) = \mathsf{true}$ holds continuously, we find all time-points T_s at which $moving(P_1, P_2) = \mathsf{true}$ is initiated, and then, for each T_s, we compute the first time-point T_e after T_s at which $moving(P_1, P_2) = \mathsf{true}$ is 'broken'. The time-points at which $F = V$ is broken are computed as follows:

$$\begin{aligned}&\mathsf{broken}(F = V,\ T_s,\ T) :- \\ &\quad \mathsf{terminatedAt}(F = V,\ T_e),\ T_s < T_e \leq T.\end{aligned} \tag{2}$$

$$\begin{aligned}&\mathsf{broken}(F = V_1,\ T_s,\ T) :- \\ &\quad \mathsf{initiatedAt}(F = V_2,\ T_e),\ T_s < T_e \leq T, \\ &\quad V_1 \neq V_2.\end{aligned} \tag{3}$$

According to rule (3), if $F = V_2$ is initiated at T_e then effectively $F = V_1$ is terminated at time T_e, for all other possible values V_1 of F. Rule (3) ensures, therefore, that a fluent cannot have more than one value at any time. We do not insist that a fluent must have a value at every time-point. In RTEC there is a difference between initiating a Boolean fluent $F = \mathsf{false}$ and terminating $F = \mathsf{true}$: the first implies, but is not implied by, the second.

Consider another example, this time from city transport management. In this application domain, transport officials are interested in computing the maximal intervals for which passenger density in buses and trams is high. This may indicate low passenger satisfaction, among others. Consider the formalisation below:

$$\begin{aligned}&\mathsf{initiatedAt}(passenger\_density(ID,\ VT) = Val,\ T) :- \\ &\quad \mathsf{happensAt}(passenger\_density\_change(ID,\ VT,\ Val),\ T).\end{aligned} \tag{4}$$

$passenger\_density\_change$ is an SDE determined from sensor data. ID concerns the vehicle of type VT (bus, tram) on which the sensors (video cameras, microphones) are mounted, while the value Val may be low, $medium$ or $high$.

Statically determined fluents In addition to the domain-independent definition of holdsFor, an event description may include domain-specific holdsFor rules, used to define the values of a fluent F in terms of the values of other fluents. We call such a fluent F *statically determined*. holdsFor rules of this kind make use of interval manipulation constructs—see the last three items of Table 1. Consider, e.g. $moving$ as in rules (1) but defined instead as a statically determined fluent:

$$\begin{aligned}&\mathsf{holdsFor}(moving(P_1, P_2) = \mathsf{true},\ I) :- \\ &\quad \mathsf{holdsFor}(walking(P_1) = \mathsf{true},\ I_1), \\ &\quad \mathsf{holdsFor}(walking(P_2) = \mathsf{true},\ I_2), \\ &\quad \mathsf{holdsFor}(close(P_1, P_2) = \mathsf{true},\ I_3), \\ &\quad \mathsf{intersect\_all}([I_1, I_2, I_3],\ I).\end{aligned} \tag{5}$$

According to the above rule, the list I of maximal intervals during which P_1 is moving with P_2 is computed by determining the list I_1 of maximal intervals during which P_1 is walking, the list I_2 of maximal intervals during which P_2 is walking, the list I_3 of maximal intervals during which P_1 is close to P_2, and then calculating the list I representing the intersections of the maximal intervals in I_1, I_2 and I_3.

RTEC provides three interval manipulation constructs: union_all, intersect_all and relative_complement_all. These are illustrated in Figure 1. The interval manipulation constructs of RTEC support the following type of definition: for all time-points T, $F = V$ holds at T if and only if some Boolean combination of fluent-value pairs holds at T. For a wide range of fluents, this is a much more concise definition than the traditional style of Event Calculus representation, i.e. identifying the various conditions under which the fluent is initiated and terminated so that maximal intervals can then be computed using the domain-independent holdsFor. Compare, e.g. the statically determined and simple fluent representations of $moving$ in rules (5) and (1) respectively.

Semantics

CE definitions in RTEC are (locally) stratified logic programs (Przymusinski 1987). We restrict attention to *hierarchical* definitions, those where it is possible to define a function *level* that maps all fluent-values $F = V$ and all events to the non-negative integers as follows. Events and statically determined fluent-values $F = V$ of level 0 are those whose happensAt and holdsFor definitions do not depend on any other events or fluents. In CE recognition, they represent the input SDEs. There are no fluent-values $F = V$ of simple fluents F in level 0. Events and simple fluent-values of level n are defined in terms of at least one event or fluent-value of level $n-1$ and a possibly empty set of events and fluent-values from levels lower than $n-1$. Statically determined fluent-values of level n are defined in terms of at least one fluent-value of level $n-1$ and a possibly empty set of fluent-values from levels lower than $n-1$. Note that fluent-values

$F = V_i$ and $F = V_j$ for $V_i \neq V_j$ could be mapped to different levels. For simplicity however, and without loss of generality, a fluent F itself is either simple or statically determined but not both.

A Simple Language

Addressing an event recognition problem in RTEC requires knowledge of Prolog, familiarisation with the Event Calculus in general, as well as understanding and memorisation of the syntactical details of RTEC in particular. In this section, we present a simple Event Calculus dialect with a high-level notation, abstracting away from technical details. This simplified Event Calculus (SimplEC) aims at supporting domain experts, such as city transport engineers and maritime patrol officials, that are not familiar with (logic) programming. SimplEC is designed to resemble simple statements in English, with some pseudo-code and mathematical elements. It does not contain obscure symbols like :−, or \+. Instead, it contains simple words like if, or, not, as well as the comma operator that indicates conjunction. There are also a few special tokens, like initiate that stem from the Event Calculus. In what follows, we illustrate the use of SimplEC through a series of examples from three real-world applications. A full account of the SimplEC statements for all these applications, as well as the grammar of SimplEC, is available at the GitHub repository of RTEC.

Recall the formalisation of passenger density from city transport management—see rule (4). This may be expressed in SimplEC as follows:

$$\begin{array}{l}\text{initiate } passenger\_density(ID, VT) = Val \text{ iff} \\ \quad passenger\_density\_change(ID, VT, Val).\end{array} \tag{6}$$

'happensAt' may be omitted since the first condition of an initiate statement is always an event. Moreover, in the absence of long-term temporal relations, the variables expressing time-points may be omitted, and 'initiatedAt' may be simply written as initiate.

The definition of $moving$ in human activity recognition, given by rule-set (1), may be written in SimplEC as follows:

$$\begin{array}{l}\text{initiate } moving(P_1, P_2) \text{ if} \\ \quad \text{start } walking(P_1), \\ \quad walking(P_2), \\ \quad close(P_1, P_2). \\ \\ \text{initiate } moving(P_1, P_2) \text{ if} \\ \quad \text{start } walking(P_2), \\ \quad walking(P_1), \\ \quad close(P_1, P_2). \\ \\ \text{initiate } moving(P_1, P_2) \text{ if} \\ \quad \text{start } close(P_1, P_2), \\ \quad walking(P_1), \\ \quad walking(P_2). \\ \\ \text{terminate } moving(P_1, P_2) \text{ if} \\ \quad \text{end } walking(P_1). \\ \\ \text{terminate } moving(P_1, P_2) \text{ if} \\ \quad \text{end } walking(P_2). \\ \\ \text{terminate } moving(P_1, P_2) \text{ if} \\ \quad \text{end } close(P_1, P_2).\end{array} \tag{7}$$

In addition to 'happensAt', 'holdsAt' is also omitted from the above statements, since after the first condition, constraints on fluents are expected (recall that $walking$ and $close$ are fluents). If more constraints on events were required, then these would have to be prefixed by 'happens'. The value of true Boolean fluents can also be omitted. To make the above formalisation even more succinct, the last three statements may be replaced by the following one:

$$\begin{array}{l}\text{terminate } moving(P_1, P_2) \text{ iff} \\ \quad \text{end } walking(P_1) \text{ or} \\ \quad \text{end } walking(P_2) \text{ or} \\ \quad \text{end } close(P_1, P_2).\end{array} \tag{8}$$

In datasets used for human activity recognition, there is no explicit information that a tracked entity is a person or an inanimate object. Therefore, in the activity definitions we try to deduce whether a tracked entity is a person or an object given, among others, the detected SDEs. We defined the fluent $person(P)$ to have value true if P has been walking, active, running or moving abruptly since P 'appeared', that is, since P was first tracked:

$$\begin{array}{l}\text{initiate } person(P) \text{ iff} \\ \quad (\text{start } walking(P) \text{ or} \\ \quad \text{start } active(P) \text{ or} \\ \quad \text{start } running(P) \text{ or} \\ \quad \text{start } abrupt(P)), \\ \quad \text{not } disappear(P). \\ \\ \text{terminate } person(P) \text{ iff} \\ \quad disappear(P).\end{array} \tag{9}$$

The value of $person(P)$ is time-dependent because the identifier P of a tracked entity that 'disappears' (is no longer tracked) at some point may be used later to refer to another entity that 'appears' (becomes tracked), and that other entity may not necessarily be a person. Note that $disappear$ is an event; however, we did not have to use the 'happens' prefix in the initiate statement. The compiler of SimplEC may deduce that $disappear$ is an event since it is used as the first condition of the 'terminate' statement.

The compiled RTEC rules are the following:

$$\begin{array}{l}\text{initiatedAt}(person(P) = \text{true},\ T) :- \\ \quad \text{happensAt}(\text{start}(walking(P) = \text{true}),\ T), \\ \quad \text{not happensAt}(disappear(P),\ T). \\ \\ \text{initiatedAt}(person(P) = \text{true},\ T) :- \\ \quad \text{happensAt}(\text{start}(active(P) = \text{true}),\ T), \\ \quad \text{not happensAt}(disappear(P),\ T). \\ \\ \text{initiatedAt}(person(P) = \text{true},\ T) :- \\ \quad \text{happensAt}(\text{start}(running(P) = \text{true}),\ T), \\ \quad \text{not happensAt}(disappear(P),\ T). \\ \\ \text{initiatedAt}(person(P) = \text{true},\ T) :- \\ \quad \text{happensAt}(\text{start}(abrupt(P) = \text{true}),\ T), \\ \quad \text{not happensAt}(disappear(P),\ T). \\ \\ \text{terminatedAt}(person(P) = \text{true},\ T) :- \\ \quad \text{happensAt}(disappear(P),\ T).\end{array} \tag{10}$$

In SimplEC, statically determined fluents are defined via statements that begin with the fluent itself. The body of

the statement consists of a number of conjunctions, disjunctions, and negations of other fluents, expressing, respectively, the intersect_all, union_all and relative_complement_all interval manipulation constructs of RTEC (see Figure 1). As an illustration, consider the statement below expressing $moving$, from the domain of activity recognition, as a statically determined fluent:

$$\begin{aligned} &moving(P_1, P_2) \text{ iff} \\ &\quad walking(P_1), \\ &\quad walking(P_2), \\ &\quad close(P_1, P_2). \end{aligned} \tag{11}$$

Recall that the corresponding definition in RTEC is given by rule (5). Note that the above statement does not include 'holdsFor' and the corresponding variables for intervals.

Another example illustrating the compactness of the statically determined fluent definitions is again taken from activity recognition. In this domain, fighting between two persons may be defined in the following way:

$$\begin{aligned} &fighting(P_1, P_2) \text{ iff} \\ &\quad (abrupt(P_1) \text{ or } abrupt(P_2)), \\ &\quad close(P_1, P_2), \\ &\quad \text{not } (inactive(P_1) \text{ or } inactive(P_2)). \end{aligned} \tag{12}$$

This statement declares that when two persons stand close to each other and at least one of them appears to be moving abruptly while none of them is inactive, then a fight is inferred to be taking place. Translating this statement into the RTEC syntax yields the following rule:

$$\begin{aligned} &\text{holdsFor}(fighting(P_1, P_2) = \text{true},\ I) :- \\ &\quad \text{holdsFor}(abrupt(P_1) = \text{true},\ I_1), \\ &\quad \text{holdsFor}(abrupt(P_2) = \text{true},\ I_2), \\ &\quad \text{union\_all}([I_1, I_2],\ I_3), \\ &\quad \text{holdsFor}(close(P_1, P_2) = \text{true},\ I_4), \\ &\quad \text{intersect\_all}([I_3, I_4],\ I_5), \\ &\quad \text{holdsFor}(inactive(P_1) = \text{true},\ I_6), \\ &\quad \text{holdsFor}(inactive(P_2) = \text{true},\ I_7), \\ &\quad \text{union\_all}([I_6, I_7],\ I_8), \\ &\quad \text{relative\_complement\_all}(I_5, [I_8], I). \end{aligned} \tag{13}$$

Rule (13) takes many more keystrokes to complete and requires the application of quite a few interval manipulation constructs, compared to statement (12). SimplEC here helps building a shorter and more natural-looking statement.

RTEC has also been used for maritime monitoring. In this domain, maritime patrol officers are interested in detecting various types of illegal, suspicious or dangerous activity, such as when a vessel is rapidly moving towards some other vessel(s). Such a behaviour could indicate a vessel pursuit or even imminent collision. Consider the formalisation below:

$$\begin{aligned} &\text{happensAt}(fastApproach(Vessel),\ T) :- \\ &\quad \text{happensAt}(speedChange(Vessel),\ T), \\ &\quad \text{holdsAt}(velocity(Vessel) = Speed,\ T), \\ &\quad Speed > 20\ knots, \\ &\quad \text{holdsAt}(coord(Vessel) = (Lon, Lat),\ T), \\ &\quad \text{not } nearPorts(Lon, Lat), \\ &\quad \text{holdsAt}(headingToVessels(Vessel) = \text{true},\ T). \end{aligned} \tag{14}$$

Unlike the specifications that we have seen so far, the activity of interest here is defined as a derived event. $speedChange(Vessel)$ is an SDE detected from vessel position signals. $velocity$ and $coord$ are fluents indicating, respectively, the speed and coordinates of a vessel. These are also given as input to RTEC. $nearPorts(Lon, Lat)$ is an atemporal predicate that becomes true when the point (Lon, Lat) is close to a port. $headingToVessels(Vessel)$ is a fluent that becomes true whenever a $Vessel$'s direction of movement is towards at least one other vessel. According to rule (14), a 'fast approach' movement is recognised when a $Vessel$ changes its speed at open sea, the new speed is above 20 knots, and there is at least one other nearby vessel towards which it is heading. The value of 20 knots was chosen by domain experts. More details about the application of RTEC to maritime monitoring may be found at (Patroumpas et al. 2017).

In addition to fluents, SimplEC supports derived events. Rule (14) e.g. may be expressed as follows:

$$\begin{aligned} &\text{happens } fastApproach(Vessel) \text{ iff} \\ &\quad speedChange(Vessel), \\ &\quad velocity(Vessel) > 20\ knots, \\ &\quad coord(Vessel) = (Lon, Lat), \\ &\quad \text{not } nearPorts(Lon, Lat), \\ &\quad headingToVessels(Vessel). \end{aligned} \tag{15}$$

Similar to initiate and terminate statements, the first condition of a happens statement is expected to be an event, while the remaining ones concern fluents unless otherwise indicated. In this example, the user has to declare separately that $nearPorts$ is an atemporal predicate. We should also like to note, in this example, the ability to condense arithmetic comparisons, such as that of lines 3 and 4 of rule (14).

Compiler

The compiler of SimplEC parses a set of statements in the simple language and, based on its grammar, translates the statements to rules in the RTEC format. Moreover, it constructs the declarations required for computing narrative assimilation queries. The declarations distinguish, for the benefit of RTEC, between simple and statically determined fluents, and between input and output entities (events and fluents). In SimplEC statement (11), for instance, the compiler will detect three statically determined fluents, namely $moving(\_, \_)$, $walking(\_)$, and $close(\_, \_)$, of which $moving(\_, \_)$ is an output entity, as it appears in the head of the statement, and the other two are input entities, as they do not appear to be defined by other, simpler entities. Subsequently, the compiler will generate the RTEC rule (5), along with the following declarations:

$$\begin{aligned} &\text{sDFluent}(moving(\_, \_)). \\ &\text{sDFluent}(walking(\_)). \\ &\text{sDFluent}(close(\_, \_)). \\ &\text{outputEntity}(moving(\_, \_)). \\ &\text{inputEntity}(walking(\_)). \\ &\text{inputEntity}(close(\_, \_)). \end{aligned} \tag{16}$$

See the GitHub repository of RTEC for example declaration files. The declarations also express the 'caching hierarchy', that is, the order in which fluents and events are

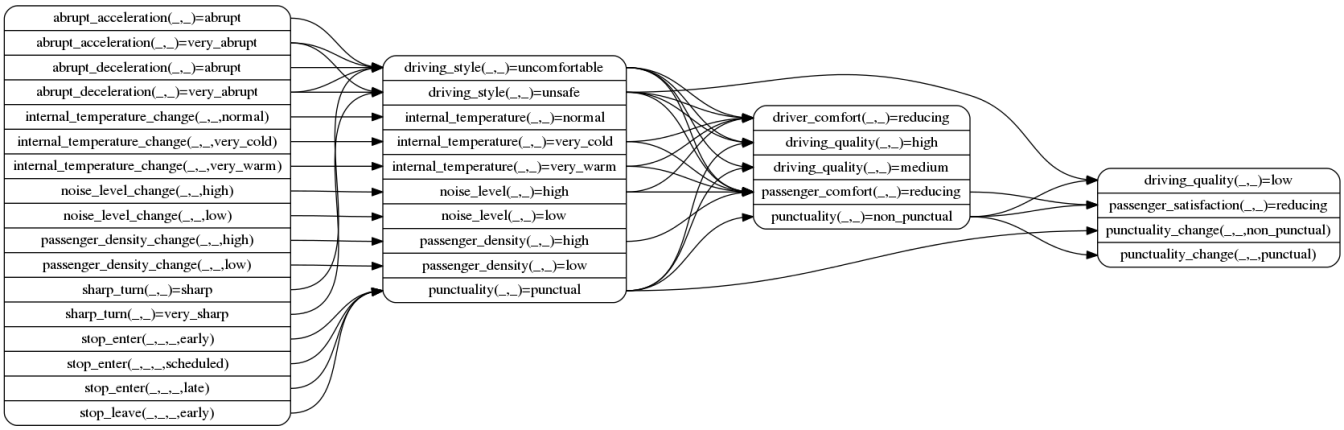


Figure 2: Dependency graph of the city transport management event description.

processed. RTEC performs bottom-up processing whereby fluents and events of level 1 of a hierarchy are processed first, subsequently moving to levels 2, 3, etc. The computed intervals of each level are cached. This way, fluent and event intervals of some level n may be simply fetched from memory when required in the processing of fluents and events of some higher level m.

To aid the user, the compiler may display the dependency graph of the event description. This is a directed graph where each vertex corresponds to a fluent or event, and for each pair of vertices (i, j) there is an edge from i to j if i appears in the body of a statement defining j. Figure 2 shows the dependency graph of an event description for city transport management. In this figure, we can observe the way in which the events and fluents affect each other. In the leftmost part of the figure there are vertices with no incoming edges. These correspond to input events and fluents that form the narrative upon which all complex activities will be recognised. In this domain, the input entities include information about the acceleration and deceleration of transport vehicles, changes in internal temperature, noise level or passenger density, as well as the time of arrival at a stop.

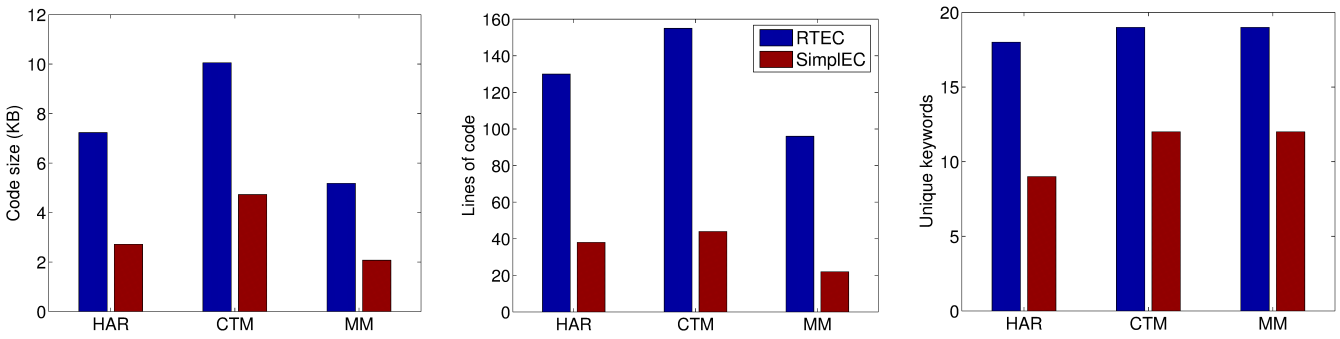
On the right of the bottom layer, there are two other layers of events and fluents that have both incoming and outgoing edges. These are output entities that also contribute to the definition of other output entities. On these layers we combine information from the bottom layer and produce higher-level information. For instance, we can recognise complex activities such as driving style and quality, vehicle punctuality and the passengers' comfort level. In the rightmost part of the figure, there is one last layer with no outgoing edges. These are the complex activities of the highest level—consider, for instance, passenger satisfaction.

Empirical Evaluation

SimplEC has been designed and implemented with a view to making the event descriptions of RTEC concise and Prolog-independent. A SimplEC user with little or no programming skills should be able to define events and the effects thereof within a particular domain, by writing short, simple and straightforward statements. To test the extent to which this conciseness is achieved by our proposed language, we compared the RTEC code with its respective SimplEC code in terms of length and simplicity in three application domains already discussed in this paper: Human activity recognition, city transport management and maritime monitoring. Figure 3 illustrates the comparison.

Figure 3a depicts the code size in bytes needed to construct equivalent formalisations in RTEC and SimplEC. In human activity recognition, the event descriptions along with the entity declarations of RTEC result in 7,406 bytes of code. The same information can be described in our proposed language using only 2,780 bytes, which corresponds to a 62.5% reduction. In city transport management, the achieved reduction is approximately 50%, while in maritime monitoring it is approximately 60%. In Figure 3b we focus the comparison on the lines of code needed in each case. As far as the human activity recognition application is concerned, 130 RTEC rules were needed in order to describe the problem. This number is brought down to just 38 using SimplEC, which corresponds to a 70.8% reduction. Similarly, for city transport management, rules were reduced by approximately 72%, while in maritime monitoring they were reduced by 77%. Finally, apart from the metrics that have to do with the brevity of the event descriptions, there are metrics that concern the simplicity of a dialect. One such metric is the amount of unique domain-independent keywords and predicates. The application of this metric to the three domains is shown in Figure 3c.

The significant difference in code size, lines of code and number of unique domain-independent predicates is mainly caused by the fact that there are several keywords, conditions and facts in the RTEC code, which are automatically inferred and generated internally by the SimplEC compiler and, therefore, need not be included in the SimplEC statement set, thus paving the way for fewer, shorter and more compact statements. These automatically inferred objects include the interval manipulation constructs and the entity declarations, among others. For instance, SimplEC statement (11) yields RTEC rule (5), along with the declarations



(a) Code size. (b) Lines of code. (c) Unique keywords.

Figure 3: Comparison between RTEC and SimplEC in three real-world applications: Human Activity Recognition (HAR), City Transport Management (CTM) and Maritime Monitoring (MM). Dark blue bars correspond to RTEC while dark red ones correspond to SimplEC.

shown in (16). Consequently, a user of SimplEC does not have to keep many special terms and domain-independent predicates in mind, thus making it easier to focus on the domain formalisation.

Related and Further Work

RTEC has a formal, declarative semantics as opposed to most complex event processing languages, several data stream processing and event query languages, and most commercial production rule systems (Cugola and Margara 2012). Moreover, RTEC supports atemporal reasoning and reasoning over background knowledge, and explicitly represents intervals, thus avoiding the related logical problems (Paschke 2005). Concerning the Event Calculus literature (e.g. (Chittaro and Montanari 1996; Cervesato and Montanari 2000; Miller and Shanahan 2002; Paschke and Bichler 2008; Artikis and Sergot 2010; Montali et al. 2013)), a key feature of RTEC is that it includes a windowing technique (Artikis, Sergot, and Paliouras 2015). In contrast, no Event Calculus system 'forgets' or represents concisely the data stream history.

We presented our efforts towards a simple language for RTEC. We observed, using real-world applications, that SimplEC helps in writing much more succinct event descriptions. However, the simple language must be evaluated by people that are not familiar with the Event Calculus. Towards this, we will make use of the domain experts of the datAcron project[1], where RTEC is used as the complex event recognition engine for maritime and aviation monitoring. We are also working towards constructing simpler and more orderly dependency graphs.

Acknowledgements

This work is funded by the H2020 project datAcron (687591).

[1] http://datacron-project.eu

References

[Artikis and Sergot 2010] Artikis, A., and Sergot, M. J. 2010. Executable specification of open multi-agent systems. *Logic Journal of the IGPL* 18(1):31–65.

[Artikis et al. 2012] Artikis, A.; Skarlatidis, A.; Portet, F.; and Paliouras, G. 2012. Logic-based event recognition. *Knowledge Eng. Review* 27(4):469–506.

[Artikis, Sergot, and Paliouras 2015] Artikis, A.; Sergot, M. J.; and Paliouras, G. 2015. An event calculus for event recognition. *IEEE Trans. Knowl. Data Eng.* 27(4):895–908.

[Bicocchi et al. 2014] Bicocchi, N.; Vassev, E.; Zambonelli, F.; and Hinchey, M. 2014. Reasoning on data streams: An approach to adaptation in pervasive systems. In *Nature of Computation and Communication - International Conference, ICTCC*, 23–32.

[Cervesato and Montanari 2000] Cervesato, I., and Montanari, A. 2000. A calculus of macro-events: Progress report. In *Seventh International Workshop on Temporal Representation and Reasoning, TIME*, 47–58.

[Chittaro and Montanari 1996] Chittaro, L., and Montanari, A. 1996. Efficient temporal reasoning in the cached event calculus. *Computational Intelligence* 12(3):359–382.

[Cugola and Margara 2012] Cugola, G., and Margara, A. 2012. Processing flows of information: From data stream to complex event processing. *ACM Computing Surveys* 44(3):15.

[Kowalski and Sergot 1986] Kowalski, R. A., and Sergot, M. J. 1986. A logic-based calculus of events. *New Generation Comput.* 4(1):67–95.

[Miller and Shanahan 2002] Miller, R., and Shanahan, M. 2002. Some alternative formulations of the event calculus. In *Computational Logic: Logic Programming and Beyond*, LNAI 2408. 452–490.

[Montali et al. 2013] Montali, M.; Maggi, F. M.; Chesani, F.; Mello, P.; and van der Aalst, W. M. P. 2013. Monitoring business constraints with the event calculus. *ACM TIST* 5(1):17:1–17:30.

[Paschke and Bichler 2008] Paschke, A., and Bichler, M. 2008. Knowledge representation concepts for automated SLA management. *Decision Support Systems* 46(1).

[Paschke 2005] Paschke, A. 2005. ECA-RuleML: An approach combining ECA rules with temporal interval-based KR event/action logics and transactional update logics. Technical report, CoRR abs/cs/0610167.

[Patroumpas et al. 2017] Patroumpas, K.; Alevizos, E.; Artikis, A.; Vodas, M.; Pelekis, N.; and Theodoridis, Y. 2017. Online event recognition from moving vessel trajectories. *GeoInformatica* 21(2):389–427.

[Przymusinski 1987] Przymusinski, T. 1987. On the declarative semantics of stratified deductive databases and logic programs. In *Foundations of Deductive Databases and Logic Programming*. Morgan.