A Distributed Event Calculus for Event Recognition

Alexandros Mavrommatis[1,2], Alexander Artikis[3,2], Anastasios Skarlatidis[2] and Georgios Paliouras[2]

Abstract. Events provide a fundamental abstraction for representing time-evolving information. Complex event recognition focuses on tracking and analysing streams of events, in order to detect patterns of special significance. The event streams may originate from various types of sensor, such as cameras and GPS sensors. Furthermore, the stream velocity and volume pose significant challenges to event processing systems. We propose dRTEC, an event recognition system that employs the Event Calculus formalism and operates in multiple processing threads. We evaluate dRTEC using two real-world applications and show that it is capable of real-time and scalable event recognition.

1 Introduction

Today's organisations need to act upon Big Data streams in order to support their resource management, capitalise on opportunities and detect threats. Towards this, event recognition systems have been particularly helpful, as they support the detection of complex events (CE)s of special significance, given streams of 'simple, derived events' (SDE)s arriving from various types of sensor [9]. A CE is a collection of events (SDEs and/or CEs) that satisfy a (spatio-)temporal pattern. In the maritime domain, for example, event 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 [2].

In previous work, we developed the 'Event Calculus for Run-Time reasoning' (RTEC), a formal computational framework for event recognition [3]. RTEC is an Event Calculus dialect [18] that includes optimisation techniques supporting efficient event recognition. A form of caching stores the results of sub-computations in the computer memory to avoid unnecessary re-computations. A simple indexing mechanism makes RTEC robust to events that are irrelevant to the computations we want to perform. A set of interval manipulation constructs simplify event patterns and improve reasoning efficiency. Furthermore, a 'windowing' mechanism makes event recognition history-independent.

RTEC is a logic programming implementation of the Event Calculus. This way, event patterns have a formal, declarative semantics [3]. On the other hand, RTEC does not have built-in support for distributed processing. This is a significant limitation, as Big Data applications, such as maritime monitoring, require the processing of high velocity SDE streams. Moreover, the integration of RTEC into non-Prolog systems is not always straightforward, requiring workarounds that hinder performance.

To deal with the increasing velocity and volume demands of today's applications, we developed 'dRTEC', a distributed implementation of RTEC. dRTEC employs Spark Streaming[4], an extension of the Apache Spark API that enables scalable, high-throughput and fault-tolerant stream processing. Reasoning in Spark Streaming may be performed exclusively in memory, where the input SDE stream is aggregated into a series of batch computations on small time intervals. dRTEC uses Spark Streaming's inherent support for distributed processing to take advantage of modern multi-core hardware for scalable event recognition.

The use of Spark Streaming additionally facilitates the integration of dRTEC, as an event recognition module, into existing (large-scale) stream processing systems. dRTEC has been evaluated in the context of two such systems. First, in the context of the SYNAISTHISI project[5], dRTEC is the human activity recognition module detecting 'long-term activities', such as fighting and leaving unattended objects, given 'short-term' activities detected on video frames by the underlying visual information processing components. In this application, we evaluated dRTEC using a benchmark activity recognition dataset. Second, in the datACRON project[6], dRTEC recognises suspicious and illegal vessel activities given a compressed vessel position stream produced by a trajectory processing module. To evaluate dRTEC on the maritime domain, we used a real position stream from over 6,000 vessels sailing through the Greek seas in the summer of 2009. The empirical analysis showed that dRTEC scales better than RTEC, both to increasing velocity SDE streams and larger numbers of CEs.

The remainder of the paper is organised as follows. In the following section we briefly review RTEC. Then, we introduce they key components of dRTEC. Section 4 presents our empirical analysis, while in Section 5 we summarise our approach, discuss related work and outline directions for further research.

2 Event Calculus for Run-Time Reasoning

dRTEC is a distributed implementation of RTEC[7], the 'Event Calculus for Run-Time reasoning' [3]. The time model of RTEC is linear including integer time-points. 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. Table 1 presents the main RTEC predicates. Variables start with an upper-case letter, while predicates and constants start with a lower-case letter. The happensAt predicate defines the event instances, the initiatedAt and termi-

[1] School of Electronic & Computer Engineering, Technical University of Crete, Greece
[2] Institute of Informatics & Telecommunications, NCSR "Demokritos", Greece
[3] Department of Maritime Studies, University of Piraeus, Greece
{blackeye,a.artikis,anskarl,paliourg}@iit.demokritos.gr

[4] http://spark.apache.org/streaming/
[5] http://iot.synaisthisi.iit.demokritos.gr/
[6] http://www.datacron-project.eu/
[7] https://github.com/aartikis/RTEC

natedAt predicates express the effects of events, while the holdsAt and holdsFor predicates express the values of the fluents. holdsAt and holdsFor are defined in such a way that, for any fluent F, holdsAt$(F=V,\ T)$ if and only if T belongs to one of the maximal intervals of I for which holdsFor$(F=V,\ I)$.

We represent instantaneous SDEs and CEs by means of happensAt, while durative SDEs and CEs are represented as fluents. The majority of CEs are durative and thus, in CE recognition the task is to compute the maximal intervals for which a fluent representing a CE has a particular value continuously.

Table 1. RTEC Predicates.

Predicate	Meaning
happensAt$(E,\ T)$	Event E occurs at time T
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
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
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

Fluents in RTEC are of two kinds: *simple* and *statically determined*. For a simple fluent F, $F=V$ holds at a particular time-point T if $F=V$ has been *initiated* by an event that has occurred at some time-point earlier than T, and has not been *terminated* in the meantime. This is an implementation of the *law of inertia*. To compute the *intervals* I for which $F=V$ holds continuously, i.e. holdsFor$(F=V,\ I)$, we compute all time-points T_s at which $F=V$ is initiated, and then, for each T_s, we find the first time-point T_f after T_s at which $F=V$ is terminated. Consider the following example from activity recognition:

$$\begin{aligned}&\text{initiatedAt}(leaving\_object(P, Obj)=\text{true},\ T) \leftarrow\\ &\quad \text{happensAt}(appear(Obj),\ T),\\ &\quad \text{holdsAt}(inactive(Obj)=\text{true},\ T),\\ &\quad \text{holdsAt}(close(P, Obj)=\text{true},\ T),\\ &\quad \text{holdsAt}(person(P)=\text{true},\ T)\end{aligned} \tag{1}$$

$$\begin{aligned}&\text{terminatedAt}(leaving\_object(P, Obj)=\text{true},\ T) \leftarrow\\ &\quad \text{happensAt}(disappear(Obj),\ T)\end{aligned} \tag{2}$$

The above rules are intended to capture the activity of leaving an object unattended. $appear$ and $disappear$ are instantaneous SDEs produced by the underlying computer vision algorithms. An entity 'appears' when it is first tracked. Similarly, an entity 'disappears' when it stops being tracked. An object carried by a person is not tracked—only the person that carries it is tracked. The object will be tracked, that is, it will 'appear', if and only if the person leaves it somewhere. $inactive$ is a durative SDE. Objects (as opposed to persons) can exhibit only inactive activity. $close(P, Obj)$ is a statically determined fluent indicating whether the distance between two entities P and Obj, tracked in the surveillance videos, is less than some threshold of pixel positions. $person(P)$ is a simple fluent indicating whether there is sufficient information that entity P is a person as opposed to an object. According to rule (1), 'leaving object' is initiated when an inactive entity starts being tracked close to a person. Rule (2) dictates that 'leaving object' stops being recognised when the entity is no longer tracked. The maximal intervals during which $leaving\_object(P, Obj)=\text{true}$ holds continuously are computed using the built-in RTEC predicate holdsFor from rules (1) and (2).

In addition to the domain-independent definition of holdsFor, RTEC supports application-dependent holdsFor rules, used to define the values of a fluent F in terms of the values of other fluents. Such a fluent F is called *statically determined*. holdsFor rules of this type make use of interval manipulation constructs—see the last three items of Table 1. Consider the following example:

$$\begin{aligned}&\text{holdsFor}(greeting(P_1, P_2)=\text{true},\ I) \leftarrow\\ &\quad \text{holdsFor}(close(P_1, P_2)=\text{true},\ I_1),\\ &\quad \text{holdsFor}(active(P_1)=\text{true},\ I_2),\\ &\quad \text{holdsFor}(inactive(P_1)=\text{true},\ I_3),\\ &\quad \text{holdsFor}(person(P_1)=\text{true},\ I_4),\\ &\quad \text{intersect\_all}([I_3, I_4],\ I_5),\\ &\quad \text{union\_all}([I_2, I_5],\ I_6),\\ &\quad \text{holdsFor}(person(P_2)=\text{true},\ I_7),\\ &\quad \text{holdsFor}(running(P_2)=\text{true},\ I_8),\\ &\quad \text{holdsFor}(abrupt(P_2)=\text{true},\ I_9),\\ &\quad \text{relative\_complement\_all}(I_7,\ [I_8, I_9],\ I_{10}),\\ &\quad \text{intersect\_all}([I_1, I_6, I_{10}],\ I)\end{aligned} \tag{3}$$

In activity recognition, we are interested in detecting whether two people are greeting each other. A greeting distinguishes meetings from other, related types of interaction. Similar to $inactive$, $active$ (mild body movement without changing location), $running$ and $abrupt$ are durative SDEs produced by the vision algorithms. According to rule (3), two tracked entities P_1 and P_2 are said to be greeting, if they are close to each other, P_1 is active or an inactive person, and P_2 is a person that is neither running nor moving abruptly.

RTEC restricts attention to *hierarchical* formalisations, those where it is possible to define a function *level* that maps all fluents and all events to the non-negative integers as follows. Events and statically determined fluents 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 simple fluents in level 0. Events and simple fluents of level n (n>0) are defined in terms of at least one event or fluent of level $n-1$ and a possibly empty set of events and fluents from levels lower than $n-1$. Statically determined fluents of level n are defined in terms of at least one fluent of level $n-1$ and a possibly empty set of fluents from levels lower than $n-1$.

RTEC performs CE recognition by means of continuous query processing, and concerns the computation of the maximal intervals of fluents. At each query time Q_i, the input entities that fall within a specified sliding window ω are taken into consideration. All input entities that took place before or at $Q_i-\omega$ are discarded/'forgotten'. This constraint ensures that the cost of CE recognition depends only on the size of ω and not on the complete SDE history. The size of ω, and the temporal distance between two consecutive query times—the 'step' Q_i-Q_{i-1}—are tuning parameters that can be chosen by the user.

When ω is longer than the step Q_i-Q_{i-1}, it is possible that an SDE occurs in the interval $(Q_i-\omega, Q_{i-1}]$ but arrives at RTEC only after Q_{i-1}; its effects are taken into account at query time Q_i. This

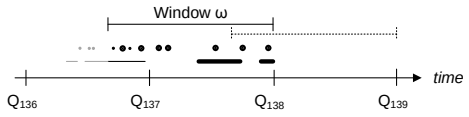


Figure 1. Windowing in RTEC.

is illustrated in Figure 1. The figure displays the occurrences of instantaneous SDEs as dots and durative ones as line segments. For CE recognition at Q_{138}, only the SDEs marked in black are considered, whereas the greyed out ones are neglected. Assume that all SDEs marked in bold arrived only after Q_{137}. Then, we observe that two SDEs were delayed i.e. they occurred before Q_{137}, but arrived only after Q_{137}. In this example, the window is larger than the step. Hence, these SDEs are not lost but considered as part of CE recognition at Q_{138}.

After 'forgetting' SDEs, RTEC computes and stores the intervals of CEs. At Q_i, the CE intervals computed by RTEC are those that can be derived from SDEs that occurred in the interval $(Q_i-\omega, Q_i]$, as recorded at time Q_i. RTEC adopts a caching technique where fluents are processed in a bottom-up manner; this way, the intervals of the fluents that are required for the processing of a fluent of level n will simply be fetched from the cache without the need for recomputation. More details about the reasoning engine of RTEC (including a complexity analysis), as well as its expressivity, may be found at [3].

3 Distributed Event Calculus

dRTEC is a distributed implementation of RTEC in Spark Streaming using the Scala programming language. Like RTEC, dRTEC performs CE recognition by means of continuous temporal projection, i.e. at each query time dRTEC computes the maximal intervals of fluents given an incoming SDE stream. Other tasks offered by other Event Calculus implementations, such as abduction, are not supported. In addition to the optimisation techniques of RTEC, such as windowing, dRTEC supports CE recognition using a structured set of operations for distributed reasoning. dRTEC follows a syntax-based, application-independent approach to translate query processing into distributed reasoning. Figure 2 illustrates the basic components of the engine using the activity recognition application. dRTEC accepts SDE streams through MQTT[8], a lightweight publish-subscribe messaging transport. Spark Streaming separates the incoming stream into individual sets, called 'micro-batches'. The window in dRTEC may contain one or more micro-batches. Each micro-batch may contain events, expressed by happensAt, and fluents, expressed by holdsFor. For example, according to the SDEs in the first micro-batch shown in Figure 2, the entity id0 started being tracked—'appeared'—at time/video frame 80. Moreover, the entity id1 was running continuously in the interval [90,100).

dRTEC performs various tasks on the incoming SDE streams. These are presented in the sections that follow. (We focus on the novel components of dRTEC, discussing only briefly the implementation of the RTEC reasoning techniques in Spark Streaming.) The CEs that are recognised using the incoming SDEs are streamed out through MQTT (see 'Output Stream' in Figure 2).

[8] http://mqtt.org/

3.1 Dynamic Grounding & Indexing

At each recognition time Q_i, RTEC grounds the CE patterns using a set of constants for the variables appearing in the patterns, except the variables related to time. Moreover, RTEC operates under the assumption that the set of constants is 'static', in the sense that it does not change over time, and known in advance. For instance, in the maritime surveillance domain, RTEC operates under the assumption that all vessel ids are known beforehand. Similarly, in activity recognition all ids of the tracked entities are assumed to be known. For many application domains, this assumption is unrealistic. More importantly, there are (many) query times in which RTEC attempts to recognise CEs for (many) constants, for which no information exists in the current window.

To address this issue, dRTEC supports 'dynamic' grounding. At each query time Q_i, dRTEC scans the SDEs of the current window ω to construct the list of entities for which CE recognition should be performed. Then, it appends to this list all entities that have CE intervals overlapping $Q_i-\omega$. Such intervals may be extended or (partially) retracted, given the information that is available in the current window. In this manner, dRTEC avoids unnecessary calculations by restricting attention to entities for which a CE may be recognised at the current query time.

Indexing is used to convert the input SDEs into a key-value pair format for data partitioning. The partitions are distributed among the available cores (processing threads) of the underlying hardware for parallel processing. Each SDE is indexed according to its entity. In activity recognition, for example, the index concerns the ids of the tracked entities (see 'Dynamic Grounding & Indexing' in Figure 2). For each window, the SDEs concerning the same entity are grouped together and subsequently sent to the same processing thread.

3.2 Non-Relational Processing

Indexing is followed by non-relational fluent processing performed at each thread in parallel (see the 'Non-Relational Processing' boxes of Figure 2). Non-relational processing refers to the computation of the maximal intervals of fluents involving a single entity. (In the absence of such fluents, dRTEC proceeds directly to 'pairing'.) In activity recognition, for example, we want to determine whether a tracked entity is a human or an object (see the rules presented in Section 2). An entity is said to be a person if it has exhibited one of the 'running', 'active', 'walking' or 'abrupt movement' short-term behaviours since it started being tracked. In other words, the classification of an entity as a person or an object depends only the short-term activities of that entity. The distinction between non-relational and relational processing allows us to trivially parallelise a significant part of the CE recognition process (non-relational CE patterns). The processing threads are independent from one another, avoiding data transfers among them that are very costly.

Non-relational, as well as relational processing, concerns both statically determined and simple fluent processing, and windowing. These tasks are discussed in Section 3.4.

3.3 Pairing & Relational Processing

Relational processing concerns CE patterns that involve two or more entities. In activity recognition, we want to recognise whether two people are moving together or fighting. Prior to relational CE recognition, dRTEC produces all possible relations that may arise from the list of entities computed by the dynamic grounding process—see 'Pairing' in Figure 2. Then, these relations are distributed to all

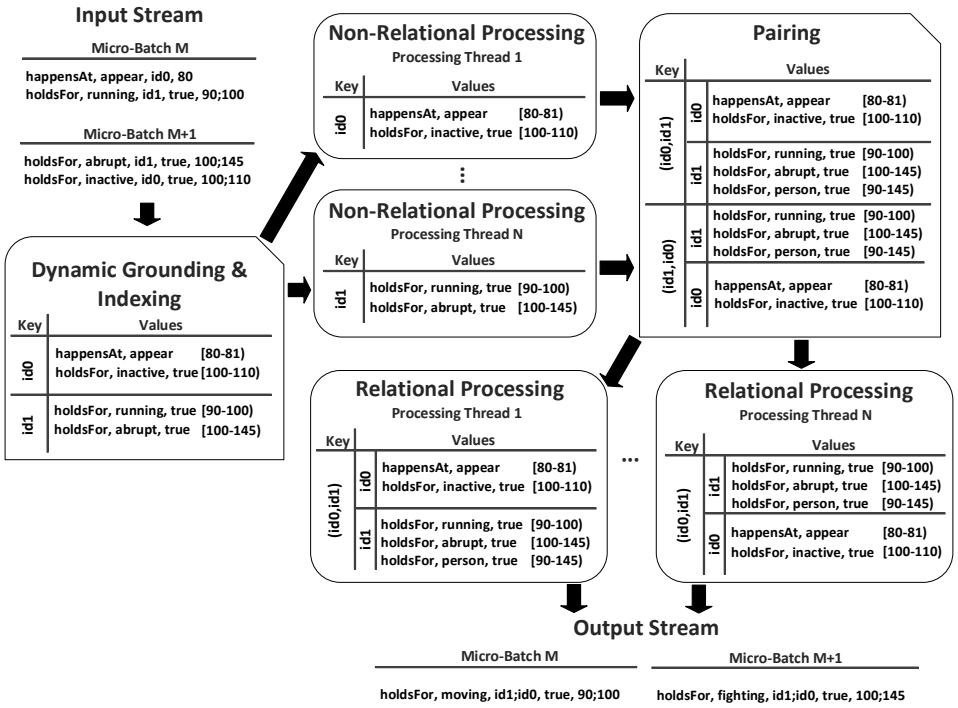


Figure 2. dRTEC processing.

available processing threads for parallel CE recognition. Note that, in contrast to non-relational processing, the information available to each processing thread is not disjoint. Assume, for example, that the pair (id0, id1) is processed by processing thread 1, while the pair (id1, id2) is processed by thread 2. Then both threads will have the output of non-relational processing concerning id1 (e.g. the list of maximal intervals during which id1 is said to be a person). However, there is no replication of computation, as the output of non-relational processing is cached, and the sets of relations of the processing threads are disjoint. Furthermore, similar to non-relational processing, each processing thread has all the necessary information, thus avoiding costly data transfers.

3.4 Fluent Processing

As mentioned earlier, both relational and non-relational processing concern the computation of the list of maximal intervals of fluents. For both types of fluent, simple and statically determined, dRTEC follows the reasoning algorithms of RTEC. For example, in the case of a simple fluent CE_s, dRTEC checks, at each query time Q_i, if there is a maximal interval of CE_s that overlaps $Q_i - \omega$. If there is such an interval then it will be discarded, while its initiating point will be kept. Then, dRTEC computes the initiating points of CE_s in $(Q_i - \omega, Q_i]$, and appends them to initiating point (if any) prior to $Q_i - \omega$. If the list of initiating points is empty then the empty list of intervals is returned. Otherwise, dRTEC computes the terminating points of CE_s in $(Q_i - \omega, Q_i]$, and pairs adjacent initiating and terminating points, as discussed in Section 2, to produce the maximal intervals.

Definitions 1 and 2 show, respectively, the initiating and terminating conditions of the 'leaving object' CE that were presented in Section 2 in the language of RTEC. Recall that 'leaving object' is a simple fluent. The GI function (GETINTERVAL, in full) retrieves the list of maximal intervals of a fluent. GI has three parameters: (a) the

Definition 1 Initiation of *leaving object* in dRTEC.

$I1 \leftarrow$ GI($occurrences$, Obj, `Fluent`($happensAt$, $appear$))
$I2 \leftarrow$ GI($occurrences$, Obj, `Fluent`($holdsFor$, $inactive$, $true$))
$I3 \leftarrow$ GI($occurrences$, (P, Obj), `Fluent`($holdsFor$, $close$, $true$))
$I4 \leftarrow$ GI($occurrences$, P, `Fluent`($holdsFor$, $person$, $true$))
$I \leftarrow I1$.INTERSECT_ALL($I2$).INTERSECT_ALL($I3$).INTERSECT_ALL($I4$)

'collection occurrences', i.e. a map pointing to the list of maximal intervals of a fluent, (b) the list of entities/arguments of the fluent, and (c) the fluent object. dRTEC uses exclusively intervals in its patterns. The occurrence of an event (e.g. 'appear') is represented by an instantaneous interval. This way, in addition to statically determined fluents, the interval manipulation constructs can be used for specifying simple fluents. In dRTEC these constructs are supported by 'interval instances' (see e.g. the last line of Definition 1).

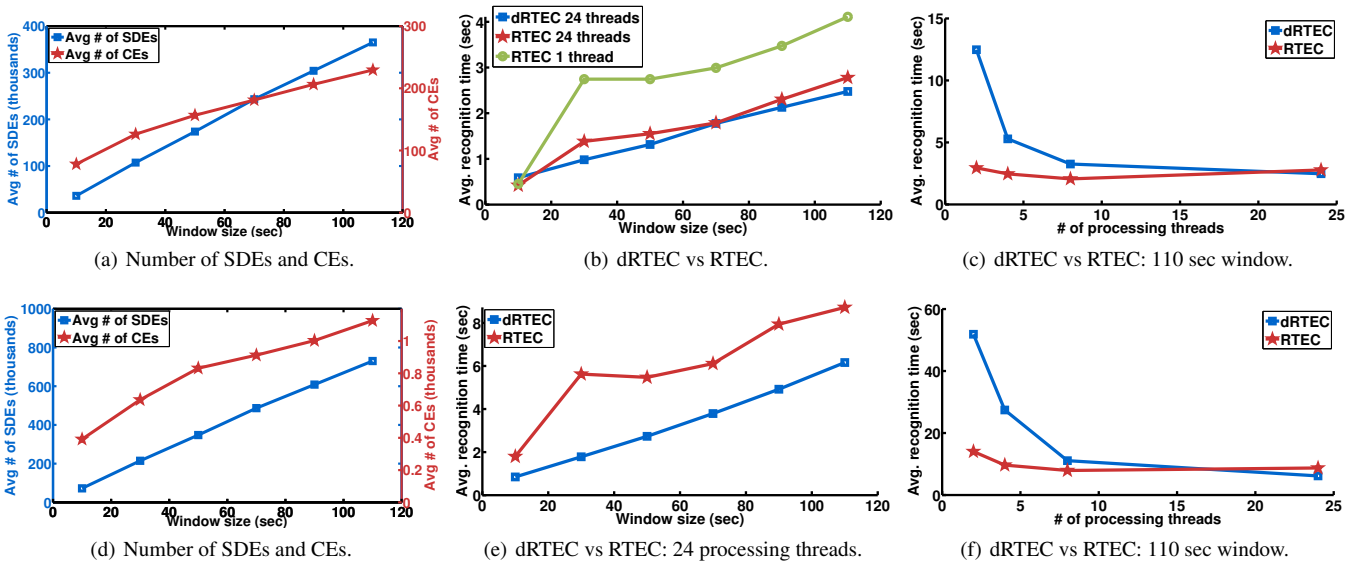
Definition 2 Termination of *leaving object* in dRTEC.

$I \leftarrow$ GI($occurrences$, Obj, `Fluent`($happensAt$, $disappear$))

Statically determined fluents in dRTEC are specified in a similar manner. Definition 3, for example, shows the specification of 'greeting' (see rule (3) for the RTEC representation).

Definition 3 Statically determined fluent *greeting* in dRTEC.

$I1 \leftarrow$ GI($occurrences$, $(P1, P2)$, `Fluent`($holdsFor$, $close$, $true$))
$I2 \leftarrow$ GI($occurrences$, $P1$, `Fluent`($holdsFor$, $active$, $true$))
$I3 \leftarrow$ GI($occurrences$, $P1$, `Fluent`($holdsFor$, $inactive$, $true$))
$I4 \leftarrow$ GI($occurrences$, $P1$, `Fluent`($holdsFor$, $person$, $true$))
$I5 \leftarrow I3$.INTERSECT_ALL($I4$)
$I6 \leftarrow I2$.UNION_ALL($I5$)
$I7 \leftarrow$ GI($occurrences$, $P2$, `Fluent`($holdsFor$, $person$, $true$))
$I8 \leftarrow$ GI($occurrences$, $P2$, `Fluent`($holdsFor$, $running$, $true$))
$I9 \leftarrow$ GI($occurrences$, $P2$, `Fluent`($holdsFor$, $abrupt$, $true$))
$I10 \leftarrow I7$.RELATIVE_COMPLEMENT_ALL($I8$.UNION_ALL($I9$))
$I \leftarrow I1$.INTERSECT_ALL($I6$).INTERSECT_ALL($I10$)



(a) Number of SDEs and CEs. (b) dRTEC vs RTEC. (c) dRTEC vs RTEC: 110 sec window.

(d) Number of SDEs and CEs. (e) dRTEC vs RTEC: 24 processing threads. (f) dRTEC vs RTEC: 110 sec window.

Figure 3. Activity recognition. Figures (a)–(c) (respectively (d)–(f)) concern the dataset with 10 (20) tracked entities.

4 Empirical Analysis

dRTEC has been evaluated in the context of two stream processing systems. In the system of the SYNAISTHISI project, dRTEC is the long-term activity recognition module operating on short-term activities detected on video frames. In the datACRON project, dRTEC recognises suspicious and illegal vessel activities given a compressed vessel position stream produced by a trajectory processing module. The empirical analysis presented below was performed on a computer with dual Intel Xeon E5-2630 processors, amounting to 24 processing threads, and 256GB RAM, running Ubuntu 14.04 LTS 64-Bit with Linux kernel 3.13 and Java OpenJDK 1.8. dRTEC is implemented in Apache Spark Streaming 1.5.2 using Scala 2.11.7. The source code, including the CE patterns for both applications, is publicly available[9]. dRTEC's warm up period is excluded from the presented results. In all cases, dRTEC recognises the same CEs as RTEC.

4.1 Activity Recognition

The SYNAISTHISI project aims at developing customisable, distributed, low-cost security and surveillance solutions. To evaluate dRTEC, we used the CAVIAR benchmark dataset[10] which consists of 28 surveillance videos of a public space. The CAVIAR videos show actors which are instructed to carry out several scenarios. Each video has been manually annotated by the CAVIAR team to provide the ground truth for activities which take place on individual video frames. These short-term activities are: entering and exiting the surveillance area, walking, running, moving abruptly, being active and being inactive. We view these activities as SDEs. The CAVIAR team has also annotated the videos with long-term activities: a person leaving an object unattended, people having a meeting, moving together, and fighting. These are the CEs that we want to recognise.

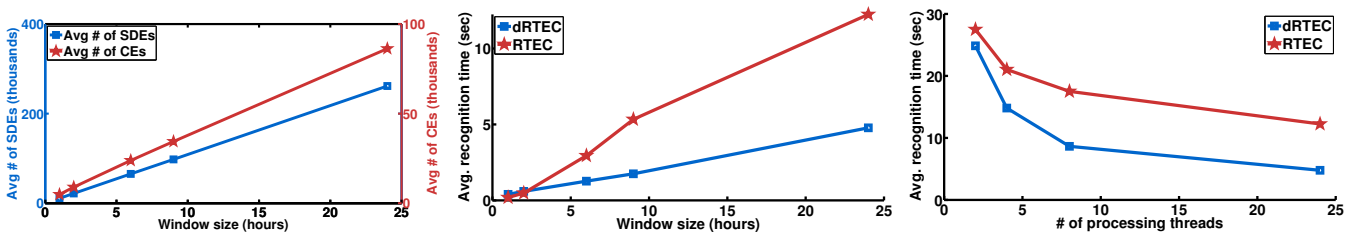
The CAVIAR dataset includes 10 tracked entities, i.e. 90 entity pairs (most CEs in this application concern a pair of entities), while the frame rate is 40 milliseconds (ms). On average, 179 SDEs are detected per second (sec). To stress test dRTEC, we constructed a larger dataset. Instead of reporting SDEs every 40 ms, the enlarged dataset provides data in every ms. The SDEs of video frame/time k of the original dataset are copied 39 times for each subsequent ms after time k. The resulting dataset has on average of 3,474 SDEs per sec. Figures 3(a)–3(c) show the experimental results on this dataset. We varied the window size from 10 sec to 110 sec. The slide step $Q_i - Q_{i-1}$ was set to be equal to the size of the window. Figure 3(a) shows the average number of SDEs per window size. The 10 sec window corresponds to approximately 36K SDEs while the 110 sec one corresponds to 365K SDEs. Figure 3(a) also shows the number of recognised CEs; these range from 80 to 230.

The average CE recognition times per window (in CPU seconds) for both dRTEC and RTEC are shown in Figure 3(b). dRTEC made use of all 24 processing threads. With the exception of the smallest window size, dRTEC outperforms RTEC. To allow for a fairer comparison, we invoked 24 instances of RTEC, each using in parallel one processing thread of the underlying hardware. Every RTEC instance was set to perform CE recognition for at most 4 entity pairs, and was provided only with the SDEs concerning the entities of these pairs (no load balancing was performed). In this setting, dRTEC outperforms RTEC for most window sizes, but only slightly.

Figure 3(c) shows the effect of increasing the number of available processing threads on the performance of dRTEC and RTEC. We varied the number of available threads from 2 to 24; the window size was set to 110 sec. RTEC achieves its best performance early—the increase of processing threads affects it only slightly. In contrast, dRTEC requires all 24 processing threads to match (slightly outperform) RTEC. The cost of data partitioning through dynamic grounding and indexing in dRTEC pays off only in the case of 24 threads.

To stress test further dRTEC, we constructed an even larger dataset by adding a copy of the previous dataset with new identifiers for the tracked entities. Thus, the resulting dataset contains a total of 20 tracked entities and 380 entity pairs, while approximately 7K SDEs take place per sec. Figures 3(d)–3(e) show the experimental results. We varied again the window size from 10 sec to 110 sec. In this case, however, the SDEs range from 72K to 730K (see Figure 3(d)). The number of recognised CEs is also much higher; it ranges from 390 to 1100.

[9] `https://github.com/blackeye42/dRTEC`
[10] http://groups.inf.ed.ac.uk/vision/CAVIAR/CAVIARDATA1



(a) Number of SDEs and CEs. (b) dRTEC vs RTEC: 24 processing threads. (c) dRTEC vs RTEC: 24 hour window.

Figure 4. Event recognition for maritime surveillance.

Figure 3(e) shows the average CE recognition times per window when all 24 processing threads were available both to dRTEC and RTEC. Each RTEC instance was set to perform CE recognition for at most 16 entity pairs, having available only the SDEs concerning the entities of these pairs. Both dRTEC and RTEC remain real-time, even in the presence of 730K SDE windows. In this set of experiments, dRTEC outperforms RTEC in all window sizes, and the difference is more significant. This is an indication that dRTEC scales better to larger datasets. Figure 3(f) shows the effect of increasing the number of processing threads. We observe a similar pattern to that of the previous experiments (see Figure 3(c)).

4.2 Maritime Surveillance

The datACRON project aims to develop novel methods for detecting threats and abnormal activity in very large numbers of moving entities operating in large geographic areas. In the stream processing system of datACRON, dRTEC serves as the component recognising various types of suspicious and illegal vessel activity. We conducted experiments against a real position stream from the Automated Identification System[11], spanning from 1 June 2009 to 31 August 2009, for 6,425 vessels sailing through the Aegean, the Ionian, and part of the Mediterranean Sea[12]. The trajectory detection module of datACRON compresses the vessel position stream to a stream of critical movement events of the following types: 'low speed', 'speed change', 'gap', indicating communication gaps, 'turn', and 'stopped', indicating that a vessel has stopped in the open sea. Each such event includes the coordinates, speed and heading of the vessel at the time of critical event detection. This way, the SDE stream includes 15,884,253 events. Given this SDE stream, we recognise the following CEs: illegal shipping, suspicious vessel delay and vessel pursuit.

We varied the window size from 1 hour, including approximately 26K SDEs, to 24 hours, including 285K SDEs (see Figure 4(a)). The slide step $Q_i - Q_{i-1}$ is always equal to the window size. The number of recognised CEs ranges from 5K to 86K. In other words, the recognised CEs are almost two orders of magnitude more than the CEs in the activity recognition application.

Figure 4(b) shows the average CE recognition times per window when all processing threads were used by both implementations. Similar to the previous experiments, each RTEC instance was given only the SDEs of the vessels for which it performs CE recognition. Although RTEC matches the performance of dRTEC for small window sizes (1 hour and 2 hour windows), dRTEC scales much better to larger window sizes. In other words, dRTEC seems to perform much better in the presence of a large number of CEs. Figure 4(c) shows the effect of increasing the processing threads. Unlike the activity recognition application, dRTEC outperforms RTEC even when just a few processing threads are available. Similar to the activity recognition domain, dRTEC makes better use of the increasing number of threads.

5 Discussion

Several techniques have been proposed in the literature for complex event processing in Big Data applications, including pattern rewriting [26], rule distribution [25], data distribution [13, 4] and parallel publish-subscribe content matching [21]. See [15, 16] for two recent surveys. Moreover, Spark Streaming has been recently used for complex event processing[13]. The key difference between our work and these approaches is the use of the Event Calculus. dRTEC inherits from RTEC the ability to represent complex temporal phenomena, explicitly represent CE intervals and thus avoid the related logical problems [23], and perform reasoning over background knowledge. This is in contrast to other complex event recognition systems, such as [8, 19], the well-known SASE engine[14] [27], and the Chronicle Recognition System [12].

Concerning the Event Calculus literature, dRTEC includes a windowing technique. On the contrary, no Event Calculus system 'forgets' or represents concisely the SDE history. Moreover, dRTEC employs a data partitioning technique using dynamic grounding and indexing. This way, dRTEC can take advantage of modern multicore hardware. This is in contrast to Event Calculus approaches [7, 5, 24, 6, 22], where the implementations have no built-in support for distributed processing. For instance, our empirical evaluation verified that dRTEC scales better than RTEC, both to SDE streams of increasing velocity and larger numbers of CEs. Note that RTEC has proven efficient enough for a variety of real-world applications [3, 2], and already outperforms the well-known Esper engine[15] in a wide range of complex event recognition tasks [1].

Several event processing systems, such as [14, 11, 8, 10, 20], operate only under the assumption that SDEs are temporally sorted. On the contrary, dRTEC supports out-of-order SDE streams and may dynamically update the intervals of recognised CEs, or recognise new CEs, as a result of delayed SDE arrival.

The use of Spark Streaming in dRTEC facilitates the integration with other modules not implemented in Prolog, such as the computer vision modules of the SYNAISTHISI project and the trajectory compression module of datACRON. The integration of RTEC with such modules was often problematic, due to issues of the libraries integrating Prolog with other programming languages. Moreover, dRTEC

[11] http://www.imo.org/OurWork/Safety/Navigation/Pages/AIS.aspx

[12] This anonymised dataset (for privacy, each vessel id has been replaced by a sequence number) is publicly available at http://chorochronos.datastories.org/?q=content/imis-3months

[13] https://github.com/Stratio/Decision

[14] http://sase.cs.umass.edu/

[15] http://www.espertech.com/esper/

avoids the memory management issues of Prolog systems that arise from continuous query computations.

For further work, we are investigating the use of a streaming infrastructure that does not rely on micro-batching (e.g. Flink[16]). Furthermore, we aim to integrate (supervised) structure learning techniques for the automated construction of Event Calculus patterns [17].

ACKNOWLEDGEMENTS

This work was funded partly by the SYNAISTHISI project, which was co-financed by the European Fund for Regional Development and from Greek National funds, and partly by the EU-funded H2020 datACRON project. We would also like to thank Elias Alevizos for his help in the empirical analysis of dRTEC.

REFERENCES

[1] E. Alevizos and A. Artikis, 'Being logical or going with the flow? A comparison of complex event processing systems', in *Proccedings of SETN*, pp. 460–474, (2014).
[2] E. Alevizos, A. Artikis, K. Patroumpas, M. Vodas, Y. Theodoridis, and N. Pelekis, 'How not to drown in a sea of information: An event recognition approach', in *IEEE International Conference on Big Data*, pp. 984–990, (2015).
[3] A. Artikis, M. Sergot, and G. Paliouras, 'An event calculus for event recognition', *Knowledge and Data Engineering, IEEE Transactions on*, **27**(4), 895–908, (2015).
[4] C. Balkesen, N. Dindar, M. Wetter, and N. Tatbul, 'Rip: Run-based intra-query parallelism for scalable complex event processing', in *Proceedings of DEBS*, (2013).
[5] I. Cervesato and A. Montanari, 'A calculus of macro-events: Progress report', in *Proceedings of TIME*, pp. 47–58, (2000).
[6] F. Chesani, P. Mello, M. Montali, and P. Torroni, 'A logic-based, reactive calculus of events', *Fundamenta Informaticae*, **105**(1-2), 135–161, (2010).
[7] L. Chittaro and A. Montanari, 'Efficient temporal reasoning in the cached event calculus', *Computational Intelligence*, **12**(3), 359–382, (1996).
[8] G. Cugola and A. Margara, 'TESLA: a formally defined event specification language', in *Proceedings of DEBS*, pp. 50–61, (2010).
[9] G. Cugola and A. Margara, 'Processing flows of information: From data stream to complex event processing', *ACM Comput. Surv.*, **44**(3), 15:1–15:62, (June 2012).
[10] N. Dindar, P. M. Fischer, M. Soner, and N. Tatbul, 'Efficiently correlating complex events over live and archived data streams', in *Proceedings of DEBS*, pp. 243–254, (2011).
[11] L. Ding, S. Chen, E. A. Rundensteiner, J. Tatemura, W.-P. Hsiung, and K. Candan, 'Runtime semantic query optimization for event stream processing', in *Proceedings of ICDE*, pp. 676–685, (2008).
[12] C. Dousson and P. Le Maigat, 'Chronicle recognition improvement using temporal focusing and hierarchisation', in *Proceedings of IJCAI*, pp. 324–329, (2007).
[13] B. Gedik, S. Schneider, M. Hirzel, and K. Wu, 'Elastic scaling for data stream processing', *IEEE Trans. Parallel Distrib. Syst.*, **25**(6), 1447–1463, (2014).
[14] D. Gyllstrom, E. Wu, H.-J. Chae, Y. Diao, P. Stahlberg, and G. Anderson, 'SASE: Complex event processing over streams', in *Proceedings of the International Conference on Innovative Data Systems Research (CIDR)*, (2007).
[15] M. Hirzel, R. Soulé, S. Schneider, B. Gedik, and R. Grimm, 'A catalog of stream processing optimizations', *ACM Comput. Surv.*, **46**(4), 46:1–46:34, (2013).
[16] Martin Hirzel, 'Partition and compose: parallel complex event processing', in *Proceedings of ACM DEBS*, pp. 191–200, (2012).
[17] N. Katzouris, A. Artikis, and G. Paliouras, 'Incremental learning of event definitions with inductive logic programming', *Machine Learning*, **100**(2-3), 555–585, (2015).

[16] `https://flink.apache.org/`

[18] R. Kowalski and M. Sergot, 'A Logic-based Calculus of Events', *New Generation Computing*, **4**(1), 67–95, (1986).
[19] J. Krämer and B. Seeger, 'Semantics and implementation of continuous sliding window queries over data streams', *ACM Transactions on Database Systems*, **34**(1), 1–49, (2009).
[20] M. Li, M. Mani, E. A. Rundensteiner, and T. Lin, 'Complex event pattern detection over streams with interval-based temporal semantics', in *Proceedings of DEBS*, pp. 291–302, (2011).
[21] A. Margara and G. Cugola, 'High-performance publish-subscribe matching using parallel hardware', *IEEE Trans. Parallel Distrib. Syst.*, **25**(1), 126–135, (2014).
[22] M. Montali, F. M. Maggi, F. Chesani, P. Mello, and W. M. P. van der Aalst, 'Monitoring business constraints with the Event Calculus', *ACM TIST*, **5**(1), (2014).
[23] A. Paschke, 'ECA-RuleML: An approach combining ECA rules with temporal interval-based KR event/action logics and transactional update logics', Technical Report 11, Technische Universität München, (2005).
[24] A. Paschke and M. Bichler, 'Knowledge representation concepts for automated SLA management', *Decision Support Systems*, **46**(1), 187–205, (2008).
[25] B. Schilling, B. Koldehofe, and K. Rothermel, 'Efficient and distributed rule placement in heavy constraint-driven event systems', in *Proceedings of IEEE HPCC*, pp. 355–364, (2011).
[26] N. P. Schultz-Møller, M. Migliavacca, and P. Pietzuch, 'Distributed complex event processing with query rewriting', in *Proceedings of DEBS*, pp. 4:1–4:12, (2009).
[27] H. Zhang, Y. Diao, and N. Immerman, 'On complexity and optimization of expensive queries in complex event processing', in *Proccedings of SIGMOD*, pp. 217–228, (2014).