Towards Automated Rule Learning for Complex Event Processing

Alessandro Margara
Dept. of Computer Science
VU University Amsterdam
a.margara@vu.nl

Gianpaolo Cugola
Dip. di Elettronica e Informazione
Politecnico di Milano, Italy
cugola@elet.polimi.it

Giordano Tamburrelli
Faculty of Informatics
Università della Svizzera Italiana, Lugano
giordano.tamburrelli@usi.ch

ABSTRACT

In CEP systems, processing takes place according to user-defined rules, which specify the relations between observed events and phenomena of interest. Writing such rules may be challenging, also for domain experts. It requires answering several questions: which events are relevant for the situation to detect and which are not? Which values should they carry? Do they need to appear in a specific, temporal order? To answer these and similar questions, we developed iCEP, a framework based on machine learning techniques to help the users in determining the hidden causality between the events received from the external environment and the situation to detect. iCEP analyzes historical traces and learns from them. It is a highly modular system, with different components considering different aspects of the rules. Depending on their knowledge of the domain, users can decide which modules to deploy and can provide hints to guide the learning process and increase its precision. In this paper, we present the design and development of iCEP in details and we evaluate its precision in a range of situations.

1. INTRODUCTION

Complex event processing (CEP) supports interpretation and combination of *primitive events* observed by a set of *sources*, into higher level *composite events* to be notified in real-time to the *sinks* in charge of reacting to them. The relationship between primitive events and the corresponding composite ones is given by a set of *rules*, which the *CEP Engine* interprets at run-time to guide its processing.

Examples of domains where such technology can be applied are sensor networks for environmental monitoring [12, 23]; financial applications requiring a continuous analysis of stocks to detect trends [18]; fraud detection tools, which observe streams of credit card transactions to prevent frauds [41]; RFID-based inventory management systems, which perform a continuous analysis of registered data to track valid paths of shipments and to capture irregularities [43]. More in general, as observed in [32], the information system of every complex company can and should be organized around an *event-based core* that acts as a nervous system to guide and control the operation of other sub-systems.

Moving from this premise, the last few years saw both the academia and the industry proposing a number of CEP engines, each one bringing its own data model, rule definition language, and processing algorithm [16]. Despite these differences, all of them suffer from an intrinsic limitation: modeling composite events in terms of rules that predicate over primitive ones is not trivial, nor human friendly, even for domain experts.

To understand the reason of this difficulty, let us consider two tipical domains for CEP: financial stock analysis and fraud detection systems. They are both characterised by massive streams of elementary data to be analysed in real-time with the goal of identifying situations of interests (e.g., a stock trend that suggests to buy or sell or a fraud happening). As a consequence, they are perfect examples where applying CEP technology. At the same time, writing rules for these settings is extremely challenging, even for domain experts. Indeed, both domains have a fundamental aspect in common: we largerly ignore the general mechanisms that guide their evolution. More specifically, we ignore the *causes* that actually trigger composite events. It is this causal relationship that we need to write the CEP rules that the engine has to interpret, and the fact that we largerly ignore it explains the difficulty in using CEP technology.

To overcome this limitation and wield all the potential hidden in CEP, we need a set of techniques to support *automated rule generation*. More specifically, we have to identify the conceptual foundations and the key algorithms that allow to *learn* from *history traces* of past primitive and composite events, the CEP rules that effectively and efficiently capture the hidden causal relationship of a given domain.

This is exactly the goal of this paper, which contributes to the research on CEP in three distinct ways. First it provides a precise but general definition of the automated rule generation problem, concretely referring to the operators of CEP languages as illustrated in [16]. Secondly, it discusses a general approach to this research challenge that builds on three fundamental pillars: decomposition into sub-problems, modularity of solutions, and ad-hoc learning algorithms. Finally, it provides a concrete initial solution to the problem: the iCEP framework. iCEP supports automated rule generation for CEP by relying on machine learning techniques to help users in determining the hidden causality between the events received from the external environment and the situation to detect. More precisely, iCEP assumes to have historical traces of events and analyzes them using *supervised learning* techniques to derive relevant CEP rules. It adopts a highly modular design, with different components considering different aspects of the rule, which allow users to decide, based on their knowledge of the domain, which modules to adopt and which hints to provide to guide the learning process and increase its precision.

The rest of the paper is organized as follows. Section 2 presents the problem statement in details, decomposes it into subproblems, and presents the general approach we propose, underlying its added-value. Section 3 provides the details of the iCEP framework, including an overview of the underpinning machine learning techniques. The performance of the iCEP prototype is studied in Section 4, highlighting our preliminary results and outlining the potential improvements. Finally, Section 5 describes related work and Section 6 provides some conclusive remarks.

2. ON AUTOMATIC RULE GENERATION

Here we introduce the problem we address in detail and describes the general approach we follow.

2.1 Events and CEP operators

The number of CEP engines and accompanying rule languages that have been proposed by industry and academia in the last few years can be roughly separated in two classes [16]. Those known as *Data Stream Management Systems* [9] offer a processing model that is similar to that of relational databases, augmented with ad-hoc operators to support on-the-fly data processing. Similar to SQL queries, the rules of DSMSs include operators that specify how to manipulate and aggregate the (streaming) input information to generate new output streams. A different model is proposed by the community working on event-based systems. The focus in this case is on detecting composite events by observing how primitive events relates in terms of content and ordering. These systems provide rule languages that allow to specify *patterns* of primitive events that trigger composite ones

The examples provided in the remainder of the paper belong to this second class, but most of the results we provide can be applied to both classes of systems. Indeed, despite their differences, it is possible to identify an abstract event model that covers most of the systems proposed so far, and a relatively small number of abstract operators that they have in common and that cover most scenarios. To put the basis for a precise definition of the problem we address, the next paragraphs present this event model and abstract operators in detail.

Event model. We assume that each event notification is characterized by a *type* and a set of *attributes*. The event type defines the number, order, names, and types of the attributes that compose the event itself. We also assume that events occur instantaneously at some points in time. Accordingly, each notification includes a *timestamp*, which represents the time of occurrence of the event it encodes. As an example, the following notification:

```
Temp@10(room=123, value=24.5)
```

captures the fact that the air temperature measured inside room 123 at time 10 was 24.5C.

CEP operators. In most CEP languages, composite events are defined using a *pattern* of primitive events. When such a pattern of events is identified the CEP engine may assume that the corresponding composite event has happened, and it may notify the interested components. As an example, the composite event `Fire` could be associated with smoke and high temperature. When smoke and high temperature notifications reach the engine it generates a `Fire` event.

To describe patterns of events, CEP languages use a number of operators. In this paper we consider five of those envisaged most relevant in [16]: selection, conjunction, sequence, window, and negation. *Selection*, filters relevant event notifications according to the values of the parameters they hold. *Conjunction*, considers patterns of event notifications composed by two or more events. *Sequence*, relates two or more event notifications that must happen in a certain order (of time). *Windows*, define the maximum timeframe of a given pattern. Finally, *negation* is used to express the fact that an event must *not* happen for the composite event to be detected.

To make this description more concrete but anyway general, here and in the remainder of the paper we use an ad-hoc, simple and intuitive syntax for event patterns, which supports the five operators above. The patterns that follow exemplify this syntax and how the five operators listed above can be used to capture an hypothetical event of `Fire`:

```
Pattern P1
within 5m. { Smoke() and Temp(value>50) }
```

```
Pattern P2
within 5m. { Smoke() and Temp(value>50) and Wind(speed>20) }
where      { Temp->Smoke, Wind->Smoke }
```

```
Pattern P3
within 5m. { Smoke() and Temp(value>50) and not Rain(mm>2) }
where      { Temp -> Smoke }
```

Pattern `P1` uses the selection operator to choose only `Temp` notifications whose value exceeds 50C, while it introduces a window of 5 minutes to find both `Smoke` and `Temp` (conjunction). Similarly, Pattern `P2` shows how the sequence operator can be used to state that both `Temp` and `Wind` must precede `Smoke` within the window of 5 minutes. Finally, Pattern `P3` shows how negation can be used to state that `Rain` must not happen within the 5 minutes window for the pattern to be detected.

Notice that in selecting event notifications we support complex *predicates* composed of a logical disjunction of *filters*, each composed of a logical conjunction of *constraints* on single attributes. As an example, the following pattern:

```
Pattern P4
Temp( (value>=50 and value<=100) or value=0)
```

selects `Temp` events that carry a temperature reading equal to 0 or included between 50 and 100.

2.2 The problem

Moving from the premises above, the problem we address can be formalized as follow: given a set of event traces Θ, and a composite event CE such that for each event trace $\varepsilon \in \Theta$ either CE is the last event in ε or it is missing from ε, derive the pattern of primitive events whose occurrence leads to CE.

As an example, from the three traces below:

```
T1: A@0, A@5, B@10, C@15, CE@15
T2: A@0, A@7, A@10, C@15
T3: A@0, B@5, B@27, C@30, CE@30
```

one could infer that CE happens when:

```
within 5s. { B() and C() }
where      { B->C }
```

This is clearly a fictional example. In practice, to derive the pattern of primitive events that is relevant for a given situation we need much more traces, as there are many aspects that must be precisely determined and the space of the solution is very large. The next section clarifies these issues and describes the general approach we propose to address them.

2.3 The general approach

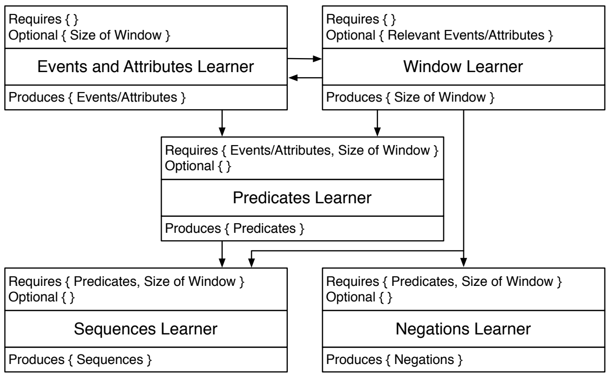
If we think at the kind of event patterns we have to identify, we may observe that the problem we try to solve, as we defined it above, can be decomposed into five sub-problems, each linked to one of the abstract operators we consider:

1. determine the relevant timeframe to consider, i.e., the window size;
2. identify the relevant event types and the relevant attributes;
3. identify the predicates that select, among the event types identified above, only those notifications that are really relevant, i.e., determine the predicates for the selection operator;
4. determine if and how relevant events are ordered within the time window, i.e., the sequences;
5. identify which event should not appear for the composite event to happen, i.e., the negated event notifications;

It is our belief that this decomposition should not remain at a logical level but it should become the high level description of the algorithm to follow. In particular, we suggest that each one of the steps above could and should be implemented into a different module of a hypothetical system for CEP rule inference. The clear separation among these steps allows the five modules that result from them to operate separately from each other, possibly using different strategies and technologies for their implementation. The next section shows a possible approach to come to this implementation, which heavily relies on machine learning to automate all the five steps, but the sharp decomposition above allows each of these steps to get different implementations for different domains, while some of them could be realized by asking a domain expert instead of relying on software. As an example, there are scenarios in which the window size is known a priori and so the first module could be eliminated altogether, while in other scenarios only relevant events are captured, thus eliminating the need for the second step. We will come back to this issue later.

3. TOWARDS A SOLUTION

The problem decomposition described in the previous section directly reflects in the modular architecture of iCEP, which is presented in Figure 1. It consists of 5 main components: (*i*) the `Events and Attributes Learner` determines which event types and which attributes inside events are relevant for the rule; (*ii*) the `Window Learner` determines the size of the temporal window that contains relevant events; (*iii*) the `Predicates Learner` infers the predicates that select relevant events based on the values of their attributes;



Requires { }
Optional { Size of Window }
Events and Attributes Learner
Produces { Events/Attributes }
Requires { }
Optional { Relevant Events/Attributes }
Window Learner
Produces { Size of Window }
Requires { Events/Attributes, Size of Window }
Optional { }
Predicates Learner
Produces { Predicates }
Requires { Predicates, Size of Window }
Optional { }
Sequences Learner
Produces { Sequences }
Requires { Predicates, Size of Window }
Optional { }
Negations Learner
Produces { Negations }

Figure 1: The High-Level Architecture of iCEP

(*iv*) the `Sequences Learner` discovers ordering relationships among primitive events; finally, the `Negations Learner` infers negations.

Figure 1 reports the input and output of each component, with arrows representing dependencies among components. For example, both the `Sequences Learner` and the `Negations Learner` depend on the `Predicates Learner`, since they rely on the set of predicates that select relevant events in order to operate. Worth noticing here is the circular dependency that exists between the `Events and Attributes Learner` and the `Window Learner`. They require each other output to perform their computation. As we will better explain later, in absence of user-provided information that may solve this dependency, we address it by letting the two components run one after the other iteratively, refining an initial hypothesis to arrive at a precise determination of both events, attributes, and windows. Notice also that in specific domains the expertise of users may suggest deploying only a subset of the components above. For instance, if the set of relevant events and attributes is known, it can be explicitly provided to the `Predicates Learner` and to the `Window Learner`, thus eliminating the need of the `Events and Attributes Learner`.

iCEP is entirely written in Java and relies on the Weka Data Mining Software 3.7.7 [26] for solving machine learning problems and generating decision trees.

3.1 Machine Learning: an Overview

Before describing the algorithms implemented inside each component of iCEP, this section briefly introduces the general concepts and the machine learning techniques it adopts.

As it name suggest, machine learning aims at building algorithms that allow to learn from data. In particular, iCEP relies on *supervised learning* techniques [28]. Generally speaking, these techniques take a set of *labeled problem instances* as their input, each problem instance being modeled as a vector of *variables*[1], and produce a set of *rules* as output, which classify instances, i.e., they map every possible instance (not only those given as input) to a certain label.

A classical example of supervised learning is the weather problem [37], which concerns the conditions that are suitable for playing a soccer game. Instances are modeled using four

[1]The exact name used in supervised learning is more often "attribute", but we use "variable" here to avoid clashes with the concept of event attribute.

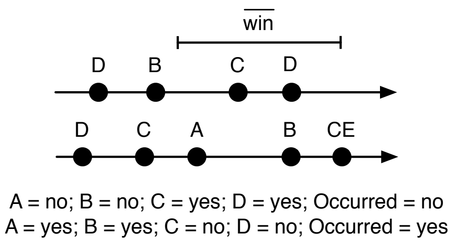

$\overline{win}$
D B C D
D C A B CE
A = no; B = no; C = yes; D = yes; Occurred = no
A = yes; B = yes; C = no; D = no; Occurred = yes

Figure 2: Events and Attributes Learner: Example of Instances Definition

variables: outlook, temperature, humidity, and windy. Labels specify if a given instance represents a situation in which playing is possible or not. The output of a machine learning technique applied to this problem is a set of rules that determines if playing given the weather conditions that hold, e.g., "if outlook = rainy and windy = true, than play = no", "if humidity = normal, then play = yes". Notice that variables may be associated with boolean values (e.g., windy), with an enumeration of atoms (e.g., outlook), or with numerical values (e.g., temperature). Finally, rules can be expressed using different formalism, e.g., using if–then statements, as exemplified above, or though decision trees.

In iCEP, each instance represents a trace of observed events. It has an associated label *Occurred*, which assumes the value *yes* if the composite event of interest occurs at the end of the trace, and *no* otherwise. Let us call *positive* instances the former, and *negative* instances the latter. The precise way in which instances are modeled (in terms of variables) depends on the iCEP component, e.g., in some cases we use variables to encode the happening of an event of a certain type, while in other case we use them to represent the value of specific attributes of occurring events.

3.2 Events and Attributes / Window Learners

As we already mentioned, the `Events and Attributes Learner` and the `Window Learner` operate in close collaboration. The former assumes known the time-span in which searching for the primitive events that caused the composite event of interest, and it searches this "window" to determine which event types (and which attributes within those types) are relevant. The latter assumes known the set of relevant events and attributes and determines the time window in which they appear.

3.2.1 Events and Attributes Learner

We start our analysis from the `Events and Attributes Learner`, assuming a lookup window $\overline{win}$. Depending on the format and semantics of primitive event notifications, we consider different variables to encode our machine learning problem.

Inference of Event Types. To illustrate our approach, we start from the simplest case, assuming that each event notification e is simply identified by its type; this means either that e does not include any attribute, or that the values of such attributes are known not to influence the occurrence of the composite event of interest. To determine which primitive events are possible causes of the composite one, we look at the event types that appear in the time window $\overline{win}$, considering both positive and negative problem instances.

More specifically, we define a different boolean variable for each event type t, which assumes the value *yes* if at least one event of type t occurred in the considered window, and *no* otherwise. Figure 2 exemplifies this approach. We consider a scenario in which 4 types of primitive events exist, namely A, B, C, and D, and we are interested in detecting the composite event CE. The top of Figure 2 shows two history traces that record past occurrences of primitive and composite events. Below, we show the two problem instances generated from such traces considering a window of size $\overline{win}$. The first one includes an event of type D and one event of type C, but no events of type A and B. It is a negative instance (labeled $Occurred = no$) because no composite event occurs at the end of the trace. On the contrary, the second trace ends with the occurrence of CE, and thus it gives rise to a positive instance (labeled $Occurred = yes$), which includes an event of type A and one event of type B, but no events of type C and D.

After generating the instances, we use them to determine which variables –i.e., which primitive event types– are relevant. To do so, we consider a measure of how informative the presence of a variable is with respect to the occurrence of the composite event of interest. In particular, we adopt the Information Gain Ratio [37] of each variable v, IGR_v, as a measure of its relevance. If the IGR_v of a variable v that represents a primitive event type t overcomes a certain threshold, we consider t as relevant for the occurrence of the composite event, otherwise we discard it.

Notice that, in principle, choosing the threshold above may seem a very critical step for a correct identification of relevant events, but in practice we found this not being the case. Indeed, while IGR may assume values between 0 and 1, in all the tests we performed (see Section 4), we found that there is a large gap between the IGR of relevant event types and that of events that are not relevant. As a consequence of this behavior, choosing a small threshold of 0.01 the `Events and Attributes Encoder` hardly ever makes any identification mistakes.

Inference of Attribute Values. In the description above, we assumed only the type of occurred events to be relevant. In many cases, however, also the values of attributes play an important role. As an example, the presence of smoke may be enough to suggest the occurrence of fire, but a temperature reading is not enough per-se. In case of temperature notifications we need to consider also the actual temperature measured (i.e., the value carried by the primitive event `Temp`) to decide if there is fire or not.

To take event attributes into consideration they have to be mapped into machine learning variables, and this mapping may assume several forms. We may aggregate the values of all events of the same type that are part of the window, for instance considering the average temperature in the window or the maximum and minimum values; or we may consider the values carried by specific events, like the last temperature reported in the window. Current version of iCEP supports some of these mappings and we plan to add more in the future.

In principle, the choice of the right mapping to use could be left to domain experts, but we may also let iCEP decide alone. Indeed, what it happens in practice is that after mapping values of event attributes into variables of problem instances, iCEP applies the same strategy used to find relevant event types, this time with the goal of determining which attribute (or aggregate of attribute values) is more relevant. So it looks at the IGR of variables to find those that are relevant. Using this approach iCEP could easily discover that both the average temperature in the time window and the last value reported are relevant, while the first value reported is not.

3.2.2 *Window Learner*

The `Window Learner` operates in a way similar to the `Events and Attributes Learner` but this time it may assume the set S of relevant event types and attributes to be known, while it has to infer the best size of window $\overline{win}$. In particular, we encode multiple machine learning problems, each time considering a different size of window win, we compute the amount of information carried by variables in S when considering each window win as:

$$I_{win} = \sum_{s \in S} IGR_s$$

and we select the window $\overline{win}$ that maximizes I_{win}, i.e.,

$$\overline{win} = \underset{win}{\arg\max} \quad I_{win}$$

In theory, this is a complex and time consuming approach since the number of possible window size to tests is virtually infinite. In practice, we observed that the function I_{win} rapidly increases around $\overline{win}$ and does not usually present local maxima. This suggests to implement a binary search approach, which adopt a finer granularity when moving close to the point of interest.

Finally, we already observed the presence of a mutual dependency between the `Types and Attributes Learner` and the `Window Learner`. In absence of external hints that could come from domain experts, we address this dependency by using an iterative approach, which allows to solve the two learning problems at the same time. In particular, for every window win considered, we use the `Types and Attributes Learner` to determine the set S of variables v such that IGR_v overcomes a certain threshold, then we use this set to calculate I_{win} through the `Window Learner`, and we repeat this cycle changing win until the value $\overline{win}$ that maximizes I_{win} is found, together with the final set S of relevant variables associated with $\overline{win}$.

3.3 Predicates Learner

In previous section we have seen how the `Events and Attributes Learner` infers which events and which attributes are worth considering. To come to this results, for each attribute of each event type, it defines one or more machine learning variables that encode such attribute (e.g., encoding the average and the maximum value of the attribute in the window) and it decides if these variables are relevant for discriminating between positive and negative instances.

The `Predicates Learner` starts from this result to infer the selection predicates that must appear in the pattern of primitive events we are looking for. In particular, for each attribute a identified as relevant, the `Predicates Learner` takes all the machine learning variables that models a, as found by the `Events and Attributes Learner`, and it builds a decision tree using the C 4.5 algorithm described in [38]. An example of decision tree for an attribute a modeled by a single variable v_a is presented in Figure 3 intermediate nodes include the value to test (v_a in our example),

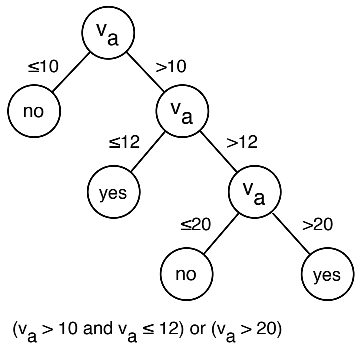


Figure 3: Predicates Learner: an Example

edges include the test condition, while leaf nodes include the value of the classification label, i.e., the occurrence of the composite event.

Starting from this decision tree, we build a predicate as follow. For each *yes* leaf, we traverse the tree up to the root, and we merge the constraints on each traversed edge, thus building a filter (i.e., a conjunct of predicates). For example, in Figure 3, the leftmost *yes* node produces the filter ($v_a >$ 10 *and* $v_a \leq 12$). The overall predicate is generated by the disjunction of such filters, each coming from a different leaf.

Finally, notice that by separately considering the impact of each attribute, we may reduce the precision of the derived results. However, this enables us to produce a single (and often compact) predicate for each attribute. On the contrary, building a single decision tree that considers *all* the relevant attributes would result in complex predicates involving constraints on multiple attributes, which could not be easily expressed by a single CEP rule. Section 4 will investigate this aspect in more detail, showing that our approach can accurately learn the expected predicates under various scenarios.

3.4 Negations Learner

With the learning steps described so far we infer which primitive events and which attribute values are "relevant" for the detection of a composite event. Indeed, this is what the IGR measures: the relevance of a given variable to discriminate between positive and negative instances. What IGR does not tell us is whether the variable is associated with positive instances or negative ones, i.e., weather the presence of the corresponding events and attributes must suggest the presence of the composite event or its absence.

The same is true for attribute values. For instance, the same decision tree in Figure 3 could result from a set of positive instances in which ($10 < v_a \leq 12$ *or* $v_a > 20$) or from a set of negative instances such that ($v_a \leq 10$ *or* $12 \leq v_a < 20$ *or* $v_a > 20$).

The `Negations Learner` discriminates between these two cases by inferring the presence of negations inside rules. It takes the lookup window $\overline{win}$, the set of relevant primitive events, and the predicates associated with relevant attributes, as input. For each relevant event e characterized by a predicate p, it counts the number of positive instances that contain e (within $\overline{win}$) and satisfy p and compares this number with the total number of positive instances. Intuitively, if this ratio is low we can deduce that e is relevant for the composite event (the IGR is known to be high) but

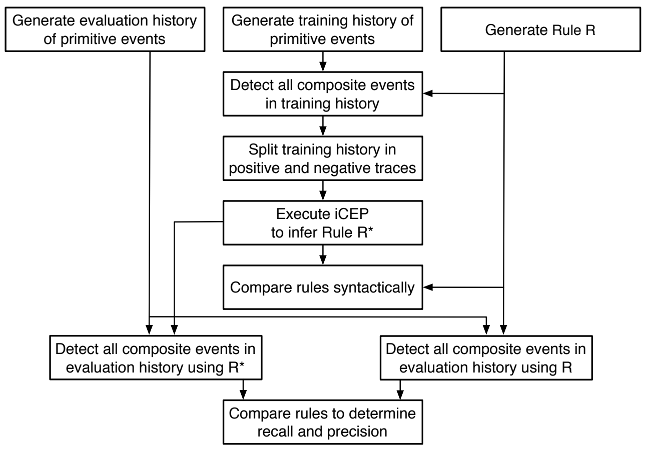

Generate evaluation history of primitive events
Generate training history of primitive events
Generate Rule R
Detect all composite events in training history
Split training history in positive and negative traces
Execute iCEP to infer Rule R*
Compare rules syntactically
Detect all composite events in evaluation history using R*
Detect all composite events in evaluation history using R
Compare rules to determine recall and precision

Figure 4: Workflow of Evaluation

it is not relevant for the composite event to happen, so it must be relevant for the composite event *not* to happen, i.e., it must appear as a negated event. Accordingly, we defined a threshold for the ratio above, when the measured ratio is below this threshold the event e must be negated. Our experiments show that a threshold of 0.9 works best in most situations.

3.5 Sequences Learner

The `Sequence Learner` is responsible for detecting the (temporal) ordering constraints that hold among primitive events. It implements an algorithm that is similar to the one described for the `Negations Learner`: it counts how many times the relevant events identified in the previous steps appear in a specific sequence within positive instances.

More specifically, for each couple of relevant primitive events e and e', taken with the associated predicates p and p', the `Sequences Learner` takes the positive instances in which both p and p' are satisfied and counts the number of those instances in which e precedes e' (at least once) and the number in which e' precedes e (at least once). If the first number is significantly higher then the second, we infer that the temporal relation $e \rightarrow e'$ exists; if the second number if significantly higher than the first, we infer the opposite relation $e' \rightarrow e$; finally, if the two numbers are similar we infer that no ordering relation exist.

4. PRELIMINARY RESULTS AND VALIDATION

This section presents some preliminary results we got using the iCEP framework, and discusses the main benefits it may bring, but also the current limitations and open issues that affect it.

4.1 Experiment Setup

As we mentioned from the very beginning, iCEP was designed as a decision support tool for the experts in charge of writing CEP rules. For this reason, in our evaluation we consider two kinds of measures: (a) we want to quantitatively determine the ability of an automatically generated rule to correctly identify composite events; (b) we aim at evaluating how valuable are the indications provided by iCEP.

Figure 4 shows the workflow we followed in our evaluation, which directly stems from these two goals. In absence of large scale case studies coming from real applications of CEP technology (which is often kept private by companies), and to better explore the parameters that can influence the behavior of iCEP, we generate primitive and composite events synthetically. More precisely, we randomly generate a *training history* of primitive events, then we define an *oracle* rule R, which we assume to perfectly represent the domain of interest, and we use R to detect all composite events in the training history. Afterwards, the training history (including primitive and composite events) is split to generate an (almost equal) number of positive and negative traces of events, as we defined them in the previous sections. These traces are the input of iCEP, which uses them to infer a rule $R*$.

To quantitatively measure the performance of iCEP–following goal (a) above–, we generate a new *evaluation history* of primitive events, and we use both R and $R*$ to detect composite events over it. This allows us to measure the *recall* of our algorithm, which is the fraction of composite events captured by R that have been also captured by $R*$; and the *precision*, which is the fraction of composite events captured by $R*$ that actually occurred, i.e., that were also captured by R.

At this point, we are also interested in evaluating subjectively how capable is iCEP at determining "correct" rules –along the lines of goal (b) above–. To do so, we compare rules R and $R*$ syntactically, to determine in which aspects they differ. We define 5 metrics for measuring the "distance" between rules R and $R*$:

- *Type*: measures the number of primitive event types that appear in R but not in $R*$, plus the number of types that appear in $R*$ but not in R;
- *Win*: measures the difference between the time window in R and that in $R*$;
- *Pred*: for each attribute part of a predicate, either in R or in $R*$, it measures the size of the intervals of values that the predicates in R and $R*$ have in common w.r.t. the entire intervals mentioned by either of the two predicates;
- *Seq*: measures the number of ordering constraints that appear in one of the two rules but not in the other. In doing so we consider the transitive closure of the ordering relationship;
- *Neg*: measures the number of negations that appear in one of the two rules but not in the other.

Notice that these metrics represent a distance between the correct rule R and the inferred rule $R*$, i.e., they represent an error in the inference process. A value of 0 represents the absence of errors.

Several parameters influence the generation of the training and evaluation histories, and of rule R. For a broad evaluation of this parameter space, we defined a default scenario, whose parameters are listed in Table 1, and we investigated the impact of each parameter separately.

In our default scenario we consider 25 different kinds of primitive events, each one having the same frequency of occurrence. Each primitive event contains a single attribute having a numeric (integer) value between 0 and 100. Rule R contains 1 sequence of three primitive events each separate

Number of event types	25
Distribution of type	Uniform
Number of attributes per event	1
Number of values per attribute	100
Number of constraints per event	1
Constraints operators distribution	=(20%),≥(40%),≤(40%)
Number of composite events	1000
Number of sequences	1
Length of sequences	3
Average window size	10
Number of negations	0

Table 1: Parameters in the Default Scenario

by a maximum of 5 time units (with a total window size of 10 units). Each primitive event is selected according to a predicate that contains one constraint. Both the training and evaluation history include one primitive event for every time unit, on average. The length of both histories is such to give rise to 1000 composite events. From the training history we derive 1000 positive traces and 1000 negative ones.

We repeated all the experiments 10 times, using different seeds to generate random values. For each experiment, we plot the average of the recall and precision we calculate and the 95% confidence interval.

4.2 Default Scenario

Recall	Precision	Type	Win	Pred	Seq	Neg
0.99	0.94	0.1	3.6	0.03	0.2	0.0

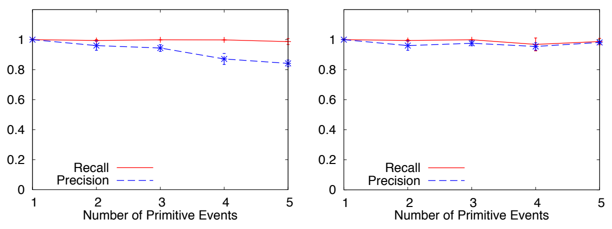
Table 2: Results of the Default Scenario

Table 2 shows the results we obtained in our default scenario. First of all, the recall is very close to 1, meaning that the rules generated by iCEP are capable of capturing almost all the composite events that occur in the evaluation history. The precision is also high and close to 0.95, meaning that the number of composite events detected are only 5% more than the composite events that actually occur. An impressive result.

For a more precise analysis of these results, we can look at the syntactic differences between Rule R to infer and Rule $R*$ generated by iCEP. First, we notice that iCEP very accurately captures the event types and predicates relevant for the composite event. Indeed, the average type distance is 0.1 over the 3 event types that are part of Rule R, which results in an percent error of 3.3%. Similarly, the average difference in predicates coverage is 0.03, meaning that the overall predicates detection error is only 3% (there are 100 different values for each attribute, see Table 1). Moreover, iCEP correctly detects the timing constraints (sequences) in almost all the tests we performed, and it always detects the absence of negations. The main difference between R and $R*$ is in the size of the window, with $R*$ overestimating it in all the tests we performed. The fact that events and attribute values are selected randomly increases the impact of overestimating the window size, which results in a (relatively) large number of false positives. In real applications, where primitive events do not happen randomly but follow a pattern related with the subsequent happening of composite events, we expect this error in estimating the size of windows to have a lower impact on precision. With reference to the example introduced in Section 2, where fire occurs when high temperature and smoke are detected within 5 minutes, increasing the size of the window should not trigger a significant number of false fire detection; indeed, if we consider the values of temperature, they are not randomly distributed, but follow precise trends, with high values being exceptional and driven by specific phenomena, like the presence of fire. The same is true for the presence of smoke.

Despite we consider iCEP as an offline tool, whose results should be used to deploy new rules into a CEP engine, we decided to measure the time it requires to process our training history to have an idea of the cost of the algorithm we propose. Using our reference hardware (a Core i7-3615QM @2.3GHz with 8GB of Ram) iCEP processes our training history (i.e., 1000 positive and 1000 negative event traces) in less than 5 min. Our experiments show that this time increases linearly with the size of windows and with the number of traces considered: even the most complex tests we will present in the remainder of this section required less than 30 min to run. Finally, we measured a maximum memory consumption of less than 1GB. These numbers allow us to conclude that neither the processing time nor the memory consumption constitute a limit for the adoption of iCEP.

4.3 Number of Primitive Events



(a) With Sequence (b) No Sequence

Figure 5: Number of Primitive Events Involved

	Type	Win	Pred	Seq	Neg
With Sequence					
1	0.0	2.0	0.0	0.0	0.0
2	0.1	5.2	0.06	0.1	0.0
3	0.1	3.6	0.03	0.2	0.0
4	0.0	1.9	0.09	0.0	0.0
5	0.0	1.1	0.04	0.0	0.0
No Sequence					
1	0.0	2.0	0.0	0.0	0.0
2	0.1	5.2	0.06	0.1	0.0
3	0.1	4.0	0.04	0.1	0.0
4	0.0	10.2	0.08	0.1	0.0
5	0.1	6.0	0.05	0.1	0.0

Table 3: Number of Primitive Events Involved

In this section we investigate how the number of primitive events in Rule R impacts the behavior of iCEP. We consider two separate scenarios: in the first one, primitive events are expected to happen one after another, in a single, long sequence, to give rise to the composite event we are looking for; in the second one, we do not require any special ordering among the primitive events, for the composite one to occur.

Figure 5 shows the precision and recall of iCEP in these two scenarios. In both cases, the recall only slightly decreases, moving from 1.00 (when considering only 1 primitive event) to 0.98 (when considering 5 primitive events). This means that Rule $R*$ captured by iCEP is capable of detecting almost all composite events that happens, even when they result from complex patterns of multiple primitive events.

Precision is also very good for the "no sequence" case, when it remains close to 1.00 independently on the number of primitive events in R (see Figure 5(b)).

Precision is worse when primitive events are organized in a specific sequence (see Figure 5(a)). At a first sight this may appear strange: indeed, by looking at Table 5, we observe that the main source of error of iCEP is represented by an overestimation of the window length and the resulting error should be bigger in absence of additional constraints (i.e., with no sequence). The phenomenon can be explained by observing that Rule R to be inferred suppose each event in the sequence being separated by at most 5 time units, i.e., it imposes a timing constraint for every sub-sequence of two events. On the contrary, iCEP produces a rule $R*$ that considers only a single time window in which all the events must occur (with no additional timing constraints among them apart from happening in order). As such, Rule $R*$ is much less restrictive than the original one, detecting much more composite events and thus lowering the precision.

Despite this difference, we notice (see Table 3, top) how the window detected by iCEP is very close to the overall window defined in R. Moreover, the difference between the two decreases with the length of the sequence.

4.4 Window Size

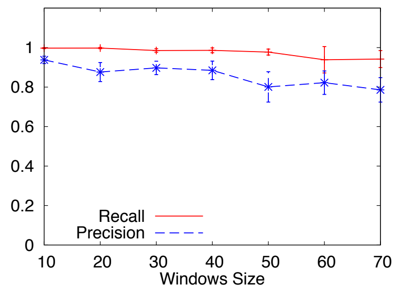


Figure 6: Window Size

	Type	Win	Pred	Seq	Neg
10	0.0	4.1	0.05	0.0	0.0
20	0.0	5.5	0.06	0.0	0.0
30	0.0	0.3	0.06	0.0	0.0
40	0.1	0.2	0.05	0.2	0.0
50	0.0	0.5	0.08	0.0	0.0
60	0.0	2.2	0.05	0.2	0.0
70	0.0	2.1	0.06	0.0	0.0

Table 4: Window Size

In this section we study the impact of the size of windows on the performance of iCEP, while keeping a single sequence with a fixed number of 3 primitive events in Rule R. The trends we observe for recall and precision (see Figure 6) are very similar to the ones in the previous section, when considering a single sequence of increasing length. Also in this case the recall is not significantly affected by the window size (moving from 0.99 to 0.94) while the precision decreases (from 0.94 to 0.78).

Again, the main difference between original Rule R and inferred Rule $R*$ is precisely in the size of window. However, this difference does not increase with the window size. Finally, it is worth mentioning that during this experiment we observed that sometimes rule $R*$ *under*estimated the actual window size.

4.5 Number of Types

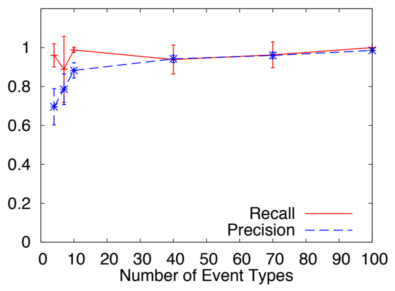


Figure 7: Number of Types

	Type	Win	Pred	Seq	Neg
4	0.1	0.3	0.11	0.2	0.0
7	0.2	1.4	0.07	0.2	0.0
10	0.0	1.5	0.06	0.0	0.0
40	0.0	6.3	0.08	0.4	0.0
70	0.0	7.2	0.04	0.2	0.0
100	0.2	5.6	0.04	0.6	0.0

Table 5: Number of Types

In this section, we investigate how the number of primitive event types part of the training history influences the results produced by iCEP. With a random distribution of types and values, having a lower number of event types results in having a higher probability that one of the types relevant for the composite event (i.e., one of the types in R) appears in a given time window. Accordingly, a lower number of types results in a higher noise, which may negatively impact the precision of iCEP.

As we can see in Figure 7, this noise only marginally impacts the average recall of iCEP. We observe, however, some fluctuations when considering a very small number of types (less then 10), which also result in larger confidence intervals. The impact on the precision is more visible: with 4 and 7 types we have a (relatively low) precision of 0.7 and 0.8, respectively.

By looking at Table 5, we notice that this result is mainly due to the errors in detecting types, predicates, and temporal constraints (sequences). Albeit being relatively small, this errors have a large impact in precision due to the low number of total types present in the training and evaluation histories.

4.6 Number of Composite Events

The number of composite events in the training history (i.e., the number of training traces) is a crucial parameter for a learning algorithm like iCEP. At the same time, this is a parameter worth investigating, since in some real situations, composite events be very rare (e.g., in emergency scenarios).

By looking at Figure 8, we observe how a very small number of composite events –and, consequently, a very small number of traces for training iCEP– results in poor recall and precision.

After 100 composite events the average recall and precision measured are above 0.8. However, different repetitions of the test show very different results. This results in large

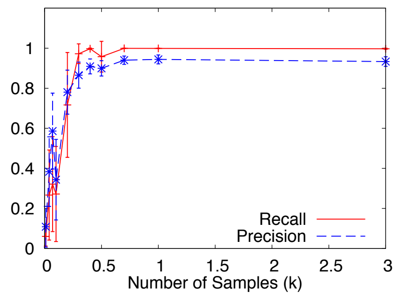


Figure 8: Number of Composite Events in Training History

	Type	Win	Pred	Seq	Neg
10	2.6	4.8	0.13	0.9	0.0
40	2.0	8.1	0.07	0.4	0.0
70	1.2	5.6	0.10	0.8	0.0
100	0.9	2.4	0.08	1.6	0.0
200	0.7	3.9	0.06	1.2	0.0
300	0.7	1.4	0.11	1.6	0.0
400	0.7	3.3	0.06	1.1	0.0
500	0.6	2.2	0.08	1.3	0.0
700	0.2	3.7	0.08	0.4	0.0
1000	0.1	3.6	0.03	0.2	0.0
3000	0.0	3.6	0.07	0.0	0.0

Table 6: Number of Composite Events in Training History

confidence intervals, but also in notable syntactical differences between Rule R and $R*$ (see Table 6): for example, iCEP makes some errors in identifying the type of primitive events involved in R and the sequences being there.

Starting from 700 composite events or more, iCEP provides good and stable results, with a recall close to 1 and a precision constantly above 0.9.

4.7 Presence of Negated Events

Recall	Precision	Type	Win	Pred	Seq	Neg
0.89	0.82	0.1	6.4	0.05	0.1	0.9

Table 7: Results in Presence of Negated Events

In this section, we study how negated events in Rule R impact the performance of iCEP. In particular, we modify the default Rule R to include a sequence of two primitive events, while a third primitive event must not appear within the window for the composite event to happen (this is a "negated" event). By looking at Table 7, we immediately observe how iCEP introduces relevant errors in the detection of negations: in 10 runs, the negation is correctly identified only once, leading to a value of 0.9 for the *Neg* metric.

This impacts the value of precision (0.82 versus 0.94 in the default scenario). Indeed, inferred Rule $R*$ detects a composite event even in presence of the negated primitive event. Moreover, the presence of negation also introduces a larger number of missing detection, with the recall moving from 0.99 of the default scenario to 0.89.

The main error iCEP makes is not considering the negated event as a relevant one: iCEP totally ignores it. We claim that the main problem here is in the way we select negative training traces, i.e., traces in which the composite event does not occur. Without any knowledge of the domain, we select them randomly. This results in a very limited number of traces in which the missing occurrence of the composite event is actually caused by the presence of the negated event. To confirm this hypothesis, we performed some additional tests providing more domain information to iCEP. In particular, if we instruct iCEP to consider the negated event as relevant for the rule (adding this information to the output of the `Events and Attributes Learner`), we observe how the remaining modules operate at best, correctly capturing all aspects of Rule R. Most importantly, with this additional information the `Negations Learner` always correctly infers that the added event must appear in a negation.

4.8 Discussion

In this section, we presented some preliminary results to validate our approach and its current implementation in the iCEP framework. In almost all the tests we performed, iCEP demonstrated very high recall and precision, even when considering long sequences and large time windows. Notice that we did not rely on any external information provided by domain experts: all the aspects of Rule R were automatically inferred.

During our tests we observed the importance of the number and quality of the traces used to train the system. With a small number of random traces (see Figure 8) iCEP exhibits low precision and recall: however, for our default scenario, 700 traces (albeit randomly chosen) were sufficient to obtain recall and precision above 0.9. Similarly, the quality of results decreases when we introduce more noise. We observed this phenomenon when reducing the number of primitive event types (see Figure 7). Along this line, a significant issue is related to the actual selection of traces. We observed this problem when considering negations: a completely random selection of negative traces leads to a poor precision. A partial knowledge of the domain is required to select traces that maximize the information required by iCEP.

Beside measuring recall and precision, we also analyzed the rules inferred by iCEP syntactically. We observed a very accurate prediction of event types and predicates, and a good detection of timing constraints. The main source of error was represented by an over-estimation of the time window, which, however, has a limited impact on precision in almost all the tests we performed.

5. RELATED WORK

This section revises related work. First, we present existing CEP systems, with particular emphasis on the languages they adopt to define rules. This aims at showing the applicability of iCEP. Second, we present existing machine learning techniques for (temporal) pattern inference and discuss how they can complement iCEP.

5.1 Complex Event Processing

The last few years have seen an increasing interest in technology for event and stream processing, and several systems have been proposed both from the academia and from the industry [32, 22].

Despite all existing solutions have been designed to accomplish the same goal, i.e., to process large volumes of flowing data on-the-fly, they present different data models and rule definition languages, as well as processing algorithms and system architectures [16].

As we already mentioned in Section 2, and following the analysis in [16], we can roughly identify two main classes of systems. On the one hand, the database community gave birth to *Data Stream Management Systems (DSMSs)* [9] to process generic information streams. On the other hand, the community working on event-based systems focused on a form of data –event notifications– with a very specific semantics, in which the time (of occurrence) plays a central role [34].

Data Stream Management Systems. DSMSs usually rely on languages derived from SQL, which specify how incoming data have to be transformed, i.e., selected, joined together, and modified, to produce one or more output streams. Processing happens in three steps [7]: first, *Stream-to-Relation* (or *windows*) operators are used to select a portion of a stream and to implicitly create traditional database tables. The actual computation occurs on these tables, using *Relation-to-Relation* –mostly standard SQL– operators. Finally, *Relation-to-Stream* operators generate new streams from tables, after data manipulation. Despite several extensions have been proposed [21, 42, 35], they all rely on the general processing schema described above.

Complex Event Processing Systems. At the opposite side of the spectrum, Complex Event Processing system are explicitly designed to capture composite events (or situations of interests) from primitive ones [22]. They interpret messages flowing into the system as notifications of events occurred in the observed world at a specific time. Rules define how composite events result from (patterns of) primitive ones.

iCEP mainly targets this class of systems, with the aim of inferring the causality relations between the presence of primitive events and the occurrence of a composite one. Nevertheless, most of the results we presented can be applied to both classes of systems.

Complex Event Processing systems often trade simplicity and performance for expressiveness, providing a reduced set of operators: for example, some languages force sequences to capture only adjacent events [11]; negations are rarely supported [31, 11] or they cannot be expressed through timing constraints [3]. Our long term goal in the development of iCEP is to exploit all the operators offered in the most expressive rule definition languages, thus enabling the derived rules to capture causal relationships present in the observed environment as precisely as possible.

Generality of iCEP. Different kinds of languages for CEP have been proposed, ranging from extensions of regular expressions [31, 11], to logic languages [6, 14, 15], and declarative languages [2, 41]. Despite their differences, by looking at existing CEP systems [16], we identified 5 main classes of operators, namely selections, conjunctions, sequences, windows, and negations.

Selection filters primitive events based on their content. As described in the previous sections, iCEP currently supports selection of both string and numeric attributes. We described and evaluated this aspect focusing on events with a single attribute. We plan to extensively study the precision and performance when generalizing to multiple attributes per event, using machine learning techniques for multi-dimentional variables, as described in Section 5.2.

Conjunction makes it possible to consider multiple events (e.g., the co-occurrence of two events *A and B* within a time window). It often include constraints on the content of attributes that can be combined together, known as *parameters* (e.g., A and B must have the same value for the x attribute). As described in Section 4, we are working on integrating new algorithms in iCEP to infer generic parameters.

Among conjunctions, *aggregates* play a central role. They enable to compute a function over a set of values (e.g., compute the average of all the values for the last 100 temperature events). As we described in Section 3, iCEP supports aggregates as part of the `Events and Attributes Learner` and `Predicates Learner` components, by explicitly encoding the value of each aggregation function of interest as an explicit Weka variable.

Another aspect related with conjunction is represented by the *selection* and *consumption policies* [14, 13]. They respectively provide answers to the following questions: in presence of multiple events satisfying all the constraints expressed for the combination, which one(s) should the system select? Which of the primitive events that led to the detection of a composite one are still valid for further detections (and which are, instead, consumed)? As discussed in Section 3, iCEP already evaluates multiple selection policies when it specifies the set of variables to use for encoding traces into machine learning instances. We plan to further explore these aspects in the next releases.

iCEP supports time windows, negations, and sequences. With respect to sequences, it is important to mention that some CEP systems adopt an interval-based, as opposed to point-based, time semantics: time is modeled using two timestamps that indicate the interval in which an event is considered valid [1]. We plan to explore the application of iCEP to this time model in the future.

Finally, some systems define iteration operators (Kleene+ [25]) to capture a priori unbounded sequences of events. This is typically exploited to detect trends (e.g., constantly increasing value of temperature). While currently not integrated in iCEP, several proposals for trend detection exist. We describe them in Section 5.2, showing how they can complement the ideas in iCEP.

Related Concepts. Other concepts have been explored in the field of Complex Event Processing that are strictly related with the idea of automatically deriving rules. Proactive event processing [20] aims at predicting the occurrence of future events, thus enabling actions that can mitigate or eliminate undesired situations.

Connected to proactive processing is the idea of computing the degree of uncertainty (often modeled as probability) associated to composite events. The first model proposed for dealing with uncertainty in CEP is described in [44], and extended in [45, 46], where the authors introduce a general framework for CEP in presence of uncertainty. Similar approaches have been studied in [39] and [19]. Finally, a tutorial on event processing under uncertainty has been presented in the DEBS (Distributed Event Based Systems) 2012 conference [8].

As a future work we plan to explore these models to understand if and how they can be combined with iCEP, to offer some indication about the confidence one can put on the derived rules and on the real occurrence of the composite events they detect.

Finally, it is important to notice that the goal of identifying composite events of interest given elementary inputs is also targeted by other approaches alternative to CEP such as neural networks [5]. However, differently from CEP these competing approaches are typically black-box solutions. Indeed, CEP rules are white-box and easily modifiable from domain experts that can check their correctness, update or even maintain them over time as soon as the domain changes. In addition, CEP is often exploited for its efficiency (compared, e.g., to neural networks), being able to process on-the-fly large volumes of input events against thousands of rules. For these reasons, in certain scenarios, CEP is preferable with respect to other competing approaches, even when such alternative solutions are more expressive and capable of capturing extremely complex situations (e.g., neural networks for face detection problems).

5.2 Machine Learning

To the best of our knowledge, iCEP represents the first attempt to adopt machine learning techniques for automated or semi-automated learning of CEP rules from historical traces.

Nevertheless, analysis, mining, and knowledge discovery from time-annotated data has been extensively studied in the past. The interested reader can refer to [40, 30] for an extensive description and classification of the solution proposed.

By following the classification presented in [40], we can organize existing solutions for temporal data mining according to two main parameters: the *object* under analysis, and the mining *paradigm*. The object under analysis can be a single *value* that changes over time, or a set of *events* that occur in time. The paradigm can be *a-priori* where the mining algorithm searches for patterns that match a user-defined template, or *classification*, where the mining algorithm clusters time sequences with similar characteristics.

The techniques adopted in iCEP are close to the discovery of event patterns through a classification paradigm [33, 36]. However, in this area, most algorithms adopt unsupervised learning techniques to detect frequent patterns or to cluster existing examples. On the contrary, iCEP guides the learning process by knowing in which points in time the composite event occurs. Moreover, existing solutions lack the expressiveness of iCEP, which is designed to automatically discover multiple parameters, including the length of a sequence, the relevant events and their values, the size of the time window, and the presence of negation.

Moreover, we plan to use use machine learning techniques for multi-dimensional variables to support events with multiple attributes. Such techniques are considered one of the most important future trends in data mining [29], and several algorithms have already been proposed [24].

Finally, we foresee the possibility to combine iCEP with algorithms for detecting trends in values. In many application fields, primitive events are produced by periodic readings of a certain value, e.g., the price of a stock, or the temperature read by a sensor. In these cases, the occurrence of a primitive event e might be meaningless, while it can be relevant to detect the presence of a specific trend in the value of e, e.g., a constantly increasing or decreasing value. Several algorithms have been proposed to detect such trends, both starting from a given template of the trend [10, 27, 4], and without any a-priori indication of the trends to be detected [47, 17].

We plan to embed these algorithms in iCEP by encoding the existence of a detected trend as a special variable of our machine learning problem.

6. CONCLUSIONS

In this paper, we addressed the problem of automated rule generation for Complex Event Processing systems. We claim that the definition of rules that precisely capture the cause-effect relationships between primitive and composite events represents a complex task, even for domain experts. A tool that guides and supports users in the definition of rules can concur in extending the adoption of CEP systems.

We precisely defined the problem and proposed a general approach for solving it. Moreover, we presented iCEP, a prototype framework that infers rules from traces of past primitive and composite event occurrences. Our approach is highly modular, enabling domain experts to adopt only a subset of components, based on the specific parts of the rule they want to infer.

An evaluation of iCEP in a wide range of scenarios has demonstrated the benefits of our approach, in terms of recall, precision, and inference of rule shape and parameters. We are currently working in extending iCEP to support additional features present in some expressive CEP language, including event selection policies and parameterization.

7. REFERENCES

[1] R. Adaikkalavan and S. Chakravarthy. SnoopIB: Interval-based event specification and detection for active databases. *Data & Knowledge Engineering*, 59(1):139 – 165, 2006.

[2] A. Adi and O. Etzion. Amit - the situation manager. *The VLDB Journal*, 13(2):177–203, 2004.

[3] J. Agrawal, Y. Diao, D. Gyllstrom, and N. Immerman. Efficient pattern matching over event streams. In *SIGMOD*, pages 147–160, New York, NY, USA, 2008. ACM.

[4] R. Agrawal, G. Psaila, E. L. Wimmers, and M. Zaït. Querying shapes of histories. In *VLDB*, pages 502–514, San Francisco, CA, USA, 1995. Morgan Kaufmann Publishers Inc.

[5] J. A. Anderson and J. Davis. *An introduction to neural networks*, volume 1. MIT Press, 1995.

[6] D. Anicic, P. Fodor, S. Rudolph, R. Stühmer, N. Stojanovic, and R. Studer. Etalis: Rule-based reasoning in event processing. *Reasoning in Event-Based Distributed Systems*, pages 99–124, 2011.

[7] A. Arasu, S. Babu, and J. Widom. The cql continuous query language: semantic foundations and query execution. *The VLDB Journal*, 15(2):121–142, 2006.

[8] A. Artikis, O. Etzion, Z. Feldman, and F. Fournier. Event processing under uncertainty. In *DEBS*, 2012.

[9] B. Babcock, S. Babu, M. Datar, R. Motwani, and J. Widom. Models and issues in data stream systems. In *PODS*, pages 1–16, New York, NY, USA, 2002. ACM.

[10] D. J. Berndt and J. Clifford. Advances in knowledge discovery and data mining. chapter Finding patterns in time series: a dynamic programming approach, pages 229–248. American Association for Artificial Intelligence, Menlo Park, CA, USA, 1996.

[11] L. Brenna, A. Demers, J. Gehrke, M. Hong, J. Ossher, B. Panda, M. Riedewald, M. Thatte, and W. White.

Cayuga: a high-performance event processing engine. In *SIGMOD*, pages 1100–1102, New York, NY, USA, 2007. ACM.

[12] K. Broda, K. Clark, R. M. 0002, and A. Russo. Sage: A logical agent-based environment monitoring and control system. In *AmI*, pages 112–117, 2009.

[13] S. Chakravarthy, V. Krishnaprasad, E. Anwar, and S.-K. Kim. Composite events for active databases: Semantics, contexts and detection. In *VLDB*, pages 606–617, San Francisco, CA, USA, 1994. Morgan Kaufmann Publishers Inc.

[14] G. Cugola and A. Margara. Tesla: a formally defined event specification language. In *DEBS*, pages 50–61, New York, NY, USA, 2010. ACM.

[15] G. Cugola and A. Margara. Complex event processing with t-rex. *Journal of Systems and Software*, 85(8):1709 – 1728, 2012.

[16] 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, 2012.

[17] G. Das, K. Lin, H. Mannila, G. Renganathan, and P. Smyth. Rule discovery from time series. *Knowledge Discovery and Data Mining*, pages 16–22, 1998.

[18] A. J. Demers, J. Gehrke, M. Hong, M. Riedewald, and W. M. White. Towards expressive publish/subscribe systems. In *EDBT*, pages 627–644, 2006.

[19] Y. Diao, B. Li, A. Liu, L. Peng, C. Sutton, T. T. L. Tran, and M. Zink. Capturing data uncertainty in high-volume stream processing. In *CIDR*, 2009.

[20] Y. Engel and O. Etzion. Towards proactive event-driven computing. In *DEBS*, pages 125–136, New York, NY, USA, 2011. ACM.

[21] Esper, http://esper.codehaus.org/, 2013.

[22] O. Etzion and P. Niblett. *Event Processing in Action.* Manning Publications Co., 2010.

[23] Event zero, http://www.eventzero.com/solutions/environment.aspx, 2012.

[24] T. Gärtner, P. A. Flach, A. Kowalczyk, and A. J. Smola. Multi-instance kernels. In *ICML*, pages 179–186, 2002.

[25] D. Gyllstrom, J. Agrawal, Y. Diao, and N. Immerman. On supporting kleene closure over event streams. In *ICDE*, pages 1391–1393, 2008.

[26] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten. The weka data mining software: an update. *SIGKDD Explor. Newsl.*, 11(1):10–18, 2009.

[27] E. Keogh and M. Pazzani. Scaling up dynamic time warping to massive datasets. *Principles of Data Mining and Knowledge Discovery*, pages 1–11, 1999.

[28] S. Kotsiantis, I. Zaharakis, and P. Pintelas. Supervised machine learning: A review of classification techniques. *Frontiers in Artificial Intelligence and Applications*, 160:3, 2007.

[29] H.-P. Kriegel, K. M. Borgwardt, P. Kröger, A. Pryakhin, M. Schubert, and A. Zimek. Future trends in data mining. *Data Mining and Knowledge Discovery*, 15(1):87–97, 2007.

[30] S. Laxman and P. Sastry. A survey of temporal data mining. *Sadhana*, 31:173–198, 2006.

[31] G. Li and H.-A. Jacobsen. Composite subscriptions in content-based publish/subscribe systems. In *Middleware*, pages 249–269. Springer-Verlag New York, Inc., 2005.

[32] D. C. Luckham. *The Power of Events: An Introduction to Complex Event Processing in Distributed Enterprise Systems.* Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2001.

[33] H. Mannila, H. Toivonen, and A. Inkeri Verkamo. Discovery of frequent episodes in event sequences. *Data Mining and Knowledge Discovery*, 1:259–289, 1997.

[34] G. Mühl, L. Fiege, and P. Pietzuch. *Distributed Event-Based Systems.* Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

[35] Oracle cep. http://www.oracle.com/technologies/soa/complex-event-processing.html, 2013.

[36] B. Padmanabhan and A. Tuzhilin. Pattern discovery in temporal databases: A temporal logic approach. In *KDD*, pages 351–354, 1996.

[37] J. Quinlan. Induction of decision trees. *Machine Learning*, 1:81–106, 1986.

[38] J. R. Quinlan. *C4.5: programs for machine learning*, volume 1. Morgan kaufmann, 1993.

[39] C. Ré, J. Letchner, M. Balazinksa, and D. Suciu. Event queries on correlated probabilistic streams. In *SIGMOD*, pages 715–728, New York, NY, USA, 2008. ACM.

[40] J. Roddick and M. Spiliopoulou. A survey of temporal knowledge discovery paradigms and methods. *IEEE Trans. on Knowledge and Data Engineering*, 14(4):750–767, 2002.

[41] N. P. Schultz-Møller, M. Migliavacca, and P. Pietzuch. Distributed complex event processing with query rewriting. In *DEB*, pages 4:1–4:12, New York, NY, USA, 2009. ACM.

[42] Streambase, http://www.streambase.com/, 2013.

[43] F. Wang and P. Liu. Temporal management of rfid data. In *VLDB*, pages 1128–1139. VLDB Endowment, 2005.

[44] S. Wasserkrug, A. Gal, and O. Etzion. A model for reasoning with uncertain rules in event composition systems. In *UAI*, pages 599–606, 2005.

[45] S. Wasserkrug, A. Gal, O. Etzion, and Y. Turchin. Complex event processing over uncertain data. In *DEBS*, pages 253–264, New York, NY, USA, 2008. ACM.

[46] S. Wasserkrug, A. Gal, O. Etzion, and Y. Turchin. Efficient processing of uncertain events in rule-based systems. *IEEE Trans. on Knowl. and Data Eng.*, 24(1):45–58, 2012.

[47] A. Weigend, F. Chen, S. Figlewski, and S. Waterhouse. Discovering technical traders in the t-bond futures market. In *KDD*, pages 354–358. Citeseer, 1998.