Detecting Temporally Related Arithmetical Patterns
An Extension of Complex Event Processing

Ronald de Haan[1] and Mikhail Roshchin[2]
[1]*Technische Universität Dresden, Dresden, Germany*
[2]*Siemens Corporate Technologies, München, Germany*

Keywords: Complex Event Processing, Diagnostic Knowledge, Temporal Relations, Arithmetical Patterns.

Abstract: When modelling diagnostic knowledge on technical systems it is often important to be able to capture complex events with a temporal structure, based on arithmetical patterns in (preprocessed) sensor data. With the current methods, such a combination is not easily possible. To solve this problem, we devise an extension of complex event processing methods, by designing a declarative language to specify the generation of events with a complex temporal structure that are based on arithmetical patterns in numerical data. This extension furthermore makes complex event processing methods more easily accessible for users that have no experience with complex event processing.

1 INTRODUCTION

In the modelling of diagnostic knowledge of large technical systems, it is important to be able to represent certain patterns in the sensor data observed in such a system. For instance, it can often be predicted on the basis of patterns in the temperature and pressure observations in a system such as a gas turbine that an expensive malfunction is about to take place. Often the patterns that are relevant for making such predictions are complex patterns, consisting of simpler patterns occurring in a particular temporal relation. There are several different existing methods to approach the problem of automatically detecting such patterns.

The method of complex event processing (CEP) (Luckham, 2001) is one way to handle complex temporal patterns of observations. Central to this method is the processing of events. An event is a message that a particular situation has occurred at a certain time point. On the one hand, events can be based on simple observations. On the other hand, complex events can be based on the occurrence of simpler events in a particular temporal structure. Complex event processing allows engineers to flexibly and declaratively specify the definition of such complex events.

Another approach to process observations, particularly in numerical data, is to deploy arithmetical methods to detect patterns in the data. This often involves preprocessing and normalization. Arithmetical methods can then be used on the preprocessed data to detect situations such as increasing or decreasing trends, or the exceedance of a threshold. The preprocessing allows us to detect such situations in the data itself, but also in derived data such as the average or standard deviation over a given time period.

1.1 Problem Statement

For expressing diagnostic knowledge of technical systems often the most meaningful patterns are complex temporally related occurrences of observations in (preprocessed) data. CEP allows us to flexibly detect the complex temporal structure, while arithmetical knowledge discovery methods allow us to detect the basic arithmetical patterns in the (preprocessed) data. However, an approach that allows us to combine the advantages of both methods is not yet available.

In this paper, we will develop a method that makes it possible to conveniently and flexibly specify complex temporally structured events based on basic arithmetical patterns in preprocessed data.

2 APPROACH

We develop a system that combines the ability of complex event processing to specify complex temporally structured events with the possibility to specify arithmetical patterns in preprocessed data.

de Haan R. and Roshchin M..
Detecting Temporally Related Arithmetical Patterns - An Extension of Complex Event Processing.
DOI: 10.5220/0004135603290332
In *Proceedings of the International Conference on Knowledge Discovery and Information Retrieval* (KDIR-2012), pages 329-332
ISBN: 978-989-8565-29-7
Copyright © 2012 SCITEPRESS (Science and Technology Publications, Lda.)

We design our system as an extension of complex event processing. We will develop a declarative language that allows engineers to express complex temporally structured patterns based on basic arithmetical patterns in the preprocessed data. Our system will then interpret expressions in this declarative language as CEP rules, that automatically generate the specified events. This allows us to leverage the existing CEP technologies.

The aim of this paper is to make it possible to specify complex events that have a temporal structure and that are based on arithmetical patterns in preprocessed, numerical data, and to do so in a declarative manner.

3 LANGUAGE EXTENSION FOR COMPLEX EVENT PROCESSING

We formally define the syntax and semantics of the language we propose. The syntax of the language is described in Figure 1. Categories in this context-free syntax are written in boldface. Note that terms of category $\mathbb{N}$ denote a natural number. We consider a term of category **user_event** as an event definition.

Let $\mathcal{X}$ and $\mathcal{D}$ be two disjoint sets of variables and data variables, respectively. We interpret terms $x(n)$ as variables $x_n \in \mathcal{X}$ and we interpret terms $d(n)$ as data variables $d_n \in \mathcal{D}$.

An event definition specifies a complex event and the conditions under which it is generated. Consider an event definition $\langle name, expr, constr \rangle$. The string *name* represents the user-defined name of the event. The conditions under which this event is generated are specified by *expr* and *constr*. The term *expr* of category **expression** is a specification of the nature of the event. This expression may contain variables of the form x_n, and data variables of the form d_n, for $n \in \mathbb{N}$. The conditions in the list *constr* specify constraints on the instantiation of such (data) variables in *expr*. More formally, a term *expr* of category **expression** specifies event generation conditions for each instantiation θ of variables and data variables in *expr* that satisfies the conditions in *constr*.

In the following, we define the conditions on instantiations θ specified by a term *constr*, and we recursively define the event generation conditions specified by a term *expr* and an instantiation θ.

3.1 Constraints on Instantiations

An instantiation θ for a term *expr* assigns each variable x_n to a natural number, and each data variable d_n to a sensor name *sensorname* of an existing sensor. This means that in the term *expr* multiple occurrences of variables x_n or data variables d_n are assigned to the same number or sensor, respectively. Furthermore, we say that an assignment θ satisfies the conditions given by *constr* if it satisfies each constraint in the list *constr*. Such constraints get the straightforward interpretation. For instance, an assignment θ satisfies a constraint $x_n \leq m$ iff $\theta(x_n) \leq m$ holds.

3.2 Recursive Definition of Events

We recursively specify the event generation conditions given by a term *expr* of category **expression**, and an instantiation θ of the (data) variables in *expr*. We can roughly divide the recursion in two parts: the definition of data events (specified by terms of category **num_data**) and the definition of intentional events (specified by terms of categories **atomic** and **expression**).

Data events represent values of (preprocessed) data. Occurrences of such events thus have a particular value, in addition to information about the nature of the event. Furthermore, data events have no duration, only a time point.

Terms of the form $data(sensorname)$ specify data events that directly represent observations of the sensor with name *sensorname*. In other words, whenever an observation for the sensor is made, such a data event should be generated. These events contain information about the observed value, the time point at which the value was observed, and the sensor for which the observation was made.

Terms of the form d_n specify similar data events for the sensor with name $\theta(d_n)$. Terms of the form $avg(data\_term, time\_term)$ specify data events as follows. For each time window of length $time\_term$, we consider all occurrences of data events specified by $data\_term$. Then, a data event is generated at the end of the time window, whose value is the arithmetic mean of the values of all considered data events. Terms of the form $sd(data\_term, time\_term)$ specify similar data events, where instead of the arithmetic mean, the standard deviation is taken as value. Terms of the form $(data\_term1 - data\_term2)$ specify data events in the following manner. For any time point in which both a data event e_1 specified by $data\_term1$ and a data event e_2 specified by $data\_term2$ occur, a data event is generated whose value is the value of e_1 minus the value of e_2. Data events are generated analogously for terms of the form $(data\_term1 + data\_term2)$, for addition instead of subtraction.

Intentional events represent (possibly complex)

time	$\rightarrow$	$\mathbb{N}$;		
var	$\rightarrow$	x($\mathbb{N}$) ;		
var_time	$\rightarrow$	**time**	$\mid$	**var** ;
data_var	$\rightarrow$	d($\mathbb{N}$) ;		
num_data	$\rightarrow$	data(**string**)	$\mid$	**data_var**
	$\mid$	sd(**num_data**,**time**)	$\mid$	avg(**num_data**,**time**)
	$\mid$	(**num_data** - **num_data**)	$\mid$	(**num_data** + **num_data**) ;
trend_dir	$\rightarrow$	up	$\mid$	down ;
threshold_value	$\rightarrow$	**num_data**	$\mid$	$\mathbb{N}$;
atomic	$\rightarrow$	trend(**trend_dir**, **num_data**, **var_time**)		
	$\mid$	less_than(**threshold_value**, **threshold_value**, **var_time**)		
	$\mid$	less_than_or_equal(**threshold_value**, **threshold_value**, **var_time**) ;		
temporal_operator	$\rightarrow$	after(**time**, **time**)	$\mid$	after
	$\mid$	during(**time**, **time**)	$\mid$	during
	$\mid$	meets(**time**)	$\mid$	meets
	$\mid$	overlaps(**time**, **time**)	$\mid$	overlaps
	$\mid$	starts(**time**)	$\mid$	starts
	$\mid$	finishes(**time**)	$\mid$	finishes ;
expression	$\rightarrow$	**atomic**		
	$\mid$	(**expression temporal_operator expression**)		
	$\mid$	(not **expression temporal_operator expression**)		
	$\mid$	(**expression temporal_operator** not **expression**) ;		
condition_list	$\rightarrow$	**condition** :: **condition_list**		
	$\mid$	[] ;		
condition	$\rightarrow$	**var** $\leq \mathbb{N}$		
	$\mid$	**var** $< \mathbb{N}$		
	$\mid$	**var** $\geq \mathbb{N}$		
	$\mid$	**var** $> \mathbb{N}$;		
user_event	$\rightarrow$	$\langle$ **string**, **expression**, **condition_list** $\rangle$;		

Figure 1: Grammar for the specification language.

patterns that are observed in the (preprocessed) data. Occurrences of such events thus have no particular value, but they do contain information about the nature of the event. Also, intentional events are interval events, i.e., they have a time point and a duration.

Terms of the form $trend(dir, data\_term, var\_time\_term)$ specify the generation of intentional events as follows. For each time window whose length equals $\theta(var\_time\_term)$, we consider all occurrences of data events specified by *data_term*. If the values of all these occurrences of data events are (chronologically) increasing, in case $dir = up$, or (chronologically) decreasing, in case $dir = down$, an event is generated for the whole time span.

A term of the form $less\_than(value\_term1, value\_term2, var\_time\_term)$ specified the generation of intentional events as follows. We distinguish several cases:

- If both *value_term*1 and *value_term*2 specify data events (as opposed to a natural number), the following generation conditions hold. For each time window whose length equals $\theta(var\_time\_term)$, we consider all moments such that both a data event e_1 specified by *value_term*1 and a data event e_2 specified by *value_term*2 occur. If for each such moment in the time window, the value of e_1 is strictly less than the value of e_2, we generate an intentional event during the whole time span.
- The case where *value_term*1 specifies a data event and *value_term*2 specifies a natural number (or vice versa) is handled similarly.
- In the case where both *value_term*1 and *value_term*2 specify natural numbers, we generate no events. This would be nonsensical.

Analogous event generation conditions are specified by terms of the form $less\_than\_or\_equal(value\_term1, value\_term2, var\_time\_term)$, with the only difference that non-strict comparison is used instead of strict comparison.

Terms of the form $(expr1\ temporal\_operator\ expr2)$ specify event generation conditions as follows. For any occurrence of an intentional event specified by $expr1$ and any occurrence of an intentional event specified by $expr2$, occurring in the temporal relation specified by $temporal\_operator$, we generate an intentional event, spanning the minimal time span in which the occurrences of both events fit.

The possible temporal operators of category **temporal_operator** get the straightforward interpretation of Allen's temporal interval operators (Allen, 1983). Temporal operators can be given extra arguments. The interpretation of temporal operators with extra arguments is as described for the corresponding operators in the documentation of Drools Fusion.

Terms of the form $(expr1\ temporal\_operator$ not $expr2)$ specify event generation conditions as follows. For any occurrence e of an intentional event specified by $expr1$ such that there is no occurrence of an intentional event specified by $expr2$, that occurs in the temporal relation, specified by $temporal\_operator$, with e, we generate an intentional event, spanning the same interval as e. Terms of the form (not $expr1\ temporal\_operator\ expr2$) specify event generation conditions analogously.

4 EXAMPLES

In order to demonstrate the working of our method, and the flexibility it offers, we will give a few examples of event definitions and the events they generate.

Consider, for instance, the case of representing diagnostic knowledge of a technical system that has several temperature sensors. Assume that we have some domain specific knowledge, that the divergence of values for two temperature sensors, possibly in combination with other events, can be a predictor for a particular malfunction in the system. We can automatically generate events corresponding to the divergence of two sensor values, for a period of at least 10 seconds, by means of the following event definition.

$$\langle \text{"distance increases for at least 10s"}, \\ trend(up,(d(1)-d(2)),x(1)),\ [x(1) \geq 10]\ \rangle$$

Assume now that we have some particular knowledge of the technical system, that a predictor for the malfunctioning behavior is that the values of two sensors differ by more than 20 units for at least 60 seconds. We can automatically generate corresponding events with the following event definition.

$$\langle \text{"distance above 20 for at least 60s"}, \\ less\_than(20,(d(1)-d(2),x(1)),\ [x(1) \geq 60]\ \rangle$$

We could use (occurrences of) those two events in a larger complex event processing system to represent diagnostic knowledge of the system. However, assume now that the sequential occurrence of the above two events is an even better predictor for the malfunctioning behavior. By means of the following event definition, we could generate corresponding events.

$$\langle \text{"complex predictor for malfunction m"}, \\ trend(up,(d(1)-d(2)),x(1))\ meets \\ less\_than(20,(d(1)-d(2),x(2))), \\ [x(1) \geq 10,\ x(2) \geq 60]\ \rangle$$

5 ADVANTAGES

Our approach, extending complex event processing with a declarative language to specify complex events based on arithmetical patterns in preprocessed data, has several advantages. First of all, it makes it possible to specify (complex) events based on arithmetical patterns in a declarative fashion, which is not possible with currently existing methods. This has the advantage that diagnostic systems dealing with such patterns become more easily maintainable and modifiable. Secondly, it allows users of complex event processing systems to intuitively and straightforwardly also take into account arithmetical patterns, in addition to the usual nominal, intentional events.

6 CONCLUSIONS

In this paper, we developed an extension of complex event processing methods that makes it possible to specify complex, temporally structured events based on arithmetical patterns in numerical data. This is done by means of a declarative language, whose expressions are interpreted as event generation conditions. This allows the automatic generation and use of events based on such declarative specifications only.

Future work would include investigating how the automatic generation of events specified with the developed declarative language could be combined with other diagnostic knowledge representation methods.

REFERENCES

Allen, J. F. (1983). Maintaining knowledge about temporal intervals. *Commun. ACM*, 26(11):832–843.

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