Contextualised Event-driven Prediction with Ontology-based Similarity

Sinan Sen
FZI Forschungszentrum Informatik
Haid-und-Neu-Straße 10-14
76131 Karlsruhe
sinan.sen@fzi.de

Jun Ma
FZI Forschungszentrum Informatik
Haid-und-Neu-Straße 10-14
76131 Karlsruhe
jun.ma@fzi.de

Abstract

Event-driven processing becomes ever important for applications such as reactive context-aware mobile applications, attention-handling and pervasive collaboration systems etc. However today's reactive systems define complex events with rather precise specifications. In some applications, such as fraud or failure detection, identification of similar event patterns may be of tremendous use. These kind of applications need to identify not only critical situations but also situations which are similar enough to them. We present a novel approach for event-driven processing which is realized by combining reactive rules with ontologies. Ontologies are used to capture the context in which certain active behavior is appropriate (i.e., to discover situations in which particular reactive rules fire). Second, ontologies together with similarity search techniques are utilised to enable discovery of similar complex event patterns.

Introduction

Event-based systems are now gaining increasing momentum as witnessed by current efforts in areas including event-driven architectures, business process management and modeling, Grid computing, Web services notifications, and message-oriented middleware. Moreover, market research companies, like Gartner[1] or Forrester[2], predict the key role of even-driven processing for making business Web application more agile.

The current approaches in event processing mainly deal with the syntactical event processing. They do not exploit a rich domain knowledge to reason about *contexts* in which a particular coplex event has occurred. As a consequence, it is not possible to detect valid *situations* in which an intelligent system should react on events. Further on, they cannot deal with uncertain and unknown events. Having background knowledge enable us to calculate similarity between coming unknown and existing known events, so that the system can react in an ad-hoc manner, i.e. on previously unknown complex events.

[1]Gartner: www.gartner.com

[2]Forrester: www.forrester.com

In this paper we propose a semantically enriched framework for modeling event-triggered reactivity and its execution. The framework is capable to monitor changes in a reactive environment and to respond to those changes appropriately. Semantic descriptions of events and contexts are used to ensure appropriate reactions on events. We formally describe complex event patterns and their corresponding contexts, which in turn allow us to identify *complex* events and reason about contexts before undertaking reactions. Additionally we are also able to discover events which are *similar* to already specified (known) events. Similarity search techniques have not been widely applied in the area of event processing yet, specially not in the contextualized manner. Contextualised and similar event processing may be of paramount importance in many real-world use cases, and we apply our event-triggered reactivity model to one of them.

The paper is organized as follows. In the second section we describe a motivating scenario. In the third section we give preliminaries and describe the state of the art in event processing and ontology-based similarity search. In the fourth section we give a new form of reactive rules, define an event calculus extended with a similarity search approach, describe the role of the context in intelligent event processing, and present a situation discrimination tree. Finally, we summarize our work and conclude in the section sixt.

Motivating Use Case Scenario

SAP's new product SAP Business ByDesign addresses small and medium enterprises. SAP Business ByDesign is delivered as Software as a Service (SaaS) over the Internet. SaaS is a software application delivery model where a software vendor develops a web-native software application and hosts and operates the application for use by its customers over the Internet. Customers do not pay for owning the software itself but rather for using it. SaaS puts new requirements like *monitoring* and *metering* to the provision of software. To ensure quality and availability of service, the hosted application instances must run reliable. In order to achieve that, all instances must be monitored. Further on, the system needs to be prophylactic to changing benchmarks, and capable to indicate forthcoming execution problems. The performance and availability of mission-critical applications are impacted

by several interconnected factors (system resources, application architecture, behavior of application code, network infrastructure, user usage etc). When problems occur, finding the bottlenecks in such a distributed system can be a complex, lengthy, and costly process. However failure detection and maintenance could be improved using Complex Event Processing (CEP) techniques. Complex event monitoring mainly informs the administrator of possible application problems. A good scalable solution is needed not only to correlate simple events to *complex events*, but also to relate complex events to their triggered *actions* and the *application contexts* in which they were issued. This will reduce significantly the event flood and will generate meaningful alerts.

The need for introducing the context in CEP can be illustrated by the following example. In order to react on a high CPU load of a CRM[3] instance (in a specified time interval) the following rules are defined: If all CRM-Monitoring-Events of the last fife minutes *(event)* exceed the threshold value of 90% for CPU consumption *(condition)* and no previous repair action happened *(context)* then do some sort of automatic healing, e.g., suspend other threads, increase or decrease the process priority etc. *(action)*. Further, the second rule says: if all CRM-Monitoring-Events of the last fife minutes *(same event)* exceed the threshold value of 90% for CPU consumption *(same condition)* and the automatic self-healing procedures did not yield any improvement from the previous situation *(different context)*, then do a different action, e.g., send a serious warning to the monitoring cockpit *(different action)*.

This example illustrates that conventional event processing based on events, conditions, and actions is not appropriate for more sophisticated real world applications. However introducing the context, we are able to react differently on the same events in different contexts (i.e., different situations). *Similarity measures* for events further enable reuse of rules to be fired in situations which were not originally specified. Similarity in event processing enable flexible monitoring in terms of handling the situations which are not completely known but similar enough to them (which is important in failure detection or fraud detection applications).

Preliminaries

Event Processing

Complex Event Processing (CEP) is a field of research concerned with task of processing multiple events, from an event cloud, with the goal of identifying the meaningful events (within the event cloud).

CEP is concerned with clouds of events, which yield only a partial *temporal* order of events. Other partial orders of interest are causality, association, taxonomy, ontology. Simple events are usually characterised by a type, their occurrence time, and optionally by a list of data parameters (which can help in detecting event patterns, or can be used in the computation after the detection). Utilising the event types, one can create complex nested expressions using the operators as in (Chakravarthy et al. 1994a): *And*, *Or*, *Sequence*, etc. Complex event specifications are patterns of events which are matched against the streams of events that occur during the run-time of the system. These patterns consist of simple event types and event operators. Simple (atomic) events are the basic events the system can detect. Complex events are detected from occurrences of one or more of atomic events. All simple events have a simple event type, which for a database application might be insert, update and delete. The types are used as placeholder in event patterns.

Early event specification languages were developed for active databases (Paton and Díaz 1989). They use complex event specifications to facilitate database triggers. One early active database system is HiPAC (McCarthy and Dayal 1989). It is an object-oriented database with transaction support. HiPAC can detect events only within a single transaction. Global event detectors are proposed which detect complex events across transaction boundaries and over longer intervals. Ode (Gehani, Jagadish, and Shmueli 1992) is another active database system with a language for the specification of event expressions. Ode proposes several basic event operators and a large amount of derived operators for ease of use and shorter syntax. The last of the classical event specification languages discussed here is Snoop (Chakravarthy et al. 1994a) and its successor SnoopIB. Snoop provides the well known operators And, Or and Sequence, as well as, Not, Any, A, A*, P, P* and Plus. A well known issue with defining an event on discrete time points instead of an interval-based semantics has been fixed later in SnoopIB (Adaikkalavan and Chakravarthy 2006).

Similarity Search

Based on the cognitive psychological approaches to similarity measure, different models have been proposed. Generally speaking, one cannot expect to find a universal measure for similarity, that can be used independently from the knowledge that is represented (Andreasen, Bulskov, and Knappe 2003). However, one approach to define a similarity measure is by using the notion of "ontology-based similarity", which helps finding elements in the domain of interest that are conceptually close but not identical.

Comparing two instances in an ontology is often based on considering the properties, the level of generality (or specificity) and the relationships with other concepts they may have. Ontology-based similarity can be further divided into different separate fragments, namely taxonomy-based similarity, feature-based similarity and similarity based on information-content (Zhang et al. 2007).

The taxonomy-based approaches calculate the similarity of terms by evaluating their position within a given taxonomy. This approach takes into account the intuitive idea that closely related terms are grouped together while distantly related terms are spaced more widely apart. Ontologies have the benefit that all terms of the domain are available as a tree structure. Since concepts are organized in a hierarchy, more general concepts are located closer near the root of the hierarchy, while more specific ones are located nearer to the leaves.

A simple metric for terms arranged as nodes in a directed

[3]CRM stands for Customer Relationship Management.

acyclic graph such as a hierarchy would be the minimal distance between the two term nodes so that similarity between two terms could be defined as the length of the shortest path between the two nodes (see also (Pedersen et al. 2007)). Alternatively, the similarity measure presented by (Wu and Palmer 1994) finds the most specific common concept that subsumes both considered concepts. On the other hand, in (Maedche et al.) has been introduced the *upwards cotopy* (UC) to measure the similarity considering their super concepts and relative places in a common hierarchy. Finally, Leacock and Chodorow define a measure based on the shortest path length between two concepts normalizing this value by twice the maximum depth of the hierarchy and smoothing the result with the logarithm.

Feature-based similarity assumes that each instance in the ontology has arbitrary properties or features. The more common features two instances have the more similar they are.

While common features tend to increase the similarity between two elements, non-common features decrease the similarity. Existing approaches calculate the similarity by combinations of shared features, distinct features, and shared absent features (Hirakawa, Xu, and Haase 1996).

Contextualised Event-triggered Reactivity with Similarity Search

System Architecture

In this section, we briefly present the architecture of our intelligent complex event processing engine and its components. The execution in event processing engines is driven by events. The event source may be an application as presented in the use case (Section Motivating Use Case Scenario) but in order to be more general the event source can be any other resource e.g., a business process, sensor or a tool). All these events are collected in the event cloud which is the input source for the intelligent complex event processing. Figure 1 shows a simplified view of our intelligent complex event processing architecture. **Event Cloud** is a collection of all received events. It contains different types of events. In a distributed scenario, where many events can be generated externally (w.r.t the core system), this component also integrates an event filter in order to preselect the relevant events. In **Event Content Analysis** the selected events from the event filter are analyzed in order to extract the event describing elements, e.g. event name, type, id, timestamp or more specific details about the occurred event. These information are forwarded to the Knowledge Base in order to map the event to the event ontology, as well as to the situation handler for the complex event processing. **Situation Handler** contains the complex event processing engine and the similarity calculation for each event. The Situation Handler detects relevant situations based on the patterns taken from the rule base and the context ontology. Depending on the context information, the *CEP Engine* recognizes different situations and fires corresponding actions. The *CEP Engine* takes also situations into account where no exact pattern can be recognized but similar ones. **Action Handler** is the task of executing actions triggered by events in well-defined contexts and situations. In general case, the action execution may change the state of a reactive system, e.g., to update a knowledge base, trigger other events, call a web service etc. For example, in the use case presented in Section Motivating Use Case Scenario, the system can execute an action to shut down some running threads which consume too much CPU power of the server. **Knowledge Base** contains the ontologies and rules for specifying events and contexts.

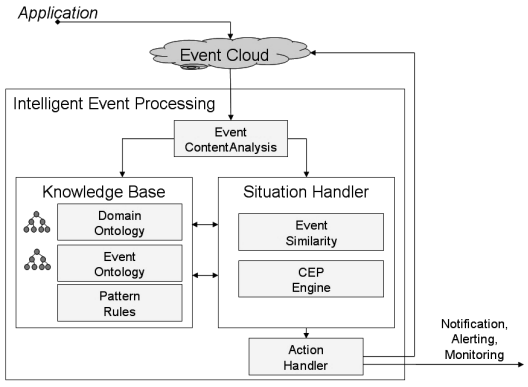

Figure 1: Intelligent Event Processing in Fraud Detection Scenario

Event and Domain Ontology We have created an event ontology to describe different types of atomic events[4]. Further on, the ontology is enriched with a number of event properties which represent event data. Purpose of the event ontology is twofold. First it is used for *complex* event detection. Secondly, the ontology is utilised for detecting *similar complex* event patterns. Figure 2 depicts the structure of an event with some defined properties. Every atomic event is related to a specific type of events in the domain ontology in order to define all types of *known* events. Known events are precisely specified events. The system knows how to detect and react upon known events. Based on the event and domain ontology we can calculate more reliable similarity between the events, either using the taxonomy of the domain ontology or the properties.

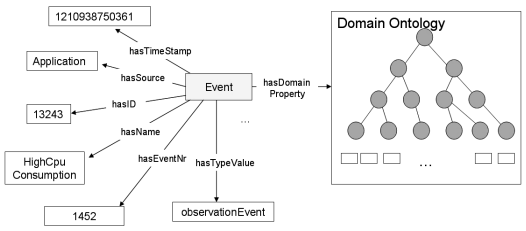

Figure 2: Event conceptualization within a domain ontology

Reactive Rules

Work on modeling *behavioral* aspects of reactive information systems originates back to the dawn of Active

[4]All ontologies from the system are accessible from http://sake.fzi.de/

Databases time (Paton and Díaz 1989). One type of reactive rules are ECA rules with typical form: "ON *event* IF *condition* DO *action*". Such a rule executes *action* as a reaction on *event*, provided that the *condition* holds. We adhere to a new form of reactive rules, i.e. ECCA (Event - Context - Condition - Action) rules; their general form is:

ON *event* WITHIN *context* IF *condition* DO *action*.

Therefore we explicitly introduce the context as an important part of reactive rules. In the standard interpretation of ECA rules, the condition part is used for representation of the contextual information. However in reality it is difficult to ensure that all relevant data for approving an automated action execution are provided in the condition part. In our opinion, an automated reactive system needs to be capable to deal with more complex contexts and situations in which data are processed. Moreover it needs to *reason* before undertaking any action (Stojanovic N 2008). Therefore we utilise the context part to find out *implicit relationships* between events and actions (and possible between other concepts as well), while the condition part is used simply to query the background information (i.e., explicitly stated knowledge[5]). For example, the context determine whether an action should be triggered as a reaction on a complex event, though it has been proved the action was not solved a problem (which initially caused that event). Having the context as tight relationship between events and actions allow us to reason before undertaking any further reactions.

Event Calculus Extended With Similarity Search

Event calculus is a mechanism for *complex* event specification and detection. Besides this standard definition of event calculus we further extend this mechanism to account for specification and detection of *similar composite* events. Therefore in this section we first define the similarity measures between events, and then specify a set of event operators considering similarity between events.

As we have already said, one aim of the intelligent complex event processing should be to detect the similarity between events in order to handle unknown event patterns. Thus the objective is to derive a function $sim(e_1, e_2)$ that measures the *distance* between the event e_1 (which is part of the specified and known event pattern) and the currently occurred event e_2. We assume that the similarity function maps instances of an ontology O into the unit interval:

$$sim(x, y) : I \times I \rightarrow [0, 1]$$

where I is the ontology instance representing possible events. The extreme value 1 occurs only if $x = y$, which is an exact match. In case the two events have nothing in common the value 0 is returned. For defining similarity in event processing, we combine more than one similarity measure, and the calculate an aggregated similarity function:

$$sim_A(e_1, e_2) := \sum_{i=1}^{n} \frac{\omega_i * sim_i(e_1, e_2)}{n}$$

Particularly we use two different types of similarity measures, i.e., similarity based on the event taxonomy sim_{tx} (Maedche et al.), and similarity based on the event properties sim_{pr}. The importance of each similarity measure in an aggregated result is captured by the parameter ω. Default value is 1, which means all similarity measures have equal importance.

In order to calculate sim_{tx} it is necessary to look into the taxonomy and identify which concepts connect e_1 and e_2. In a simple way this can be considered as the shortest path between the two event instances.

$$sim_{tx}(e_1, e_2) = min(\sum_{path_{e_1}}^{n} \sum_{path_{e_2}}^{m} \frac{|UCpath_{e_1}(e_1) \cap UCpath_{e_2}(e_2)|}{|UCpath_{e_1}(e_1) \cup UCpath_{e_2}(e_2)|})$$

This measure is able to handle different connections between the two instances and to select the shortest one if many paths are available. One drawback of the shortest path calculation is the increase of complexity, since we consider all possible paths in the taxonomy between two instances. However, since the similarity calculation of events is executed at design time, it has no influence on the overall event processing performance at run time. For the calculation of similarity the reachable superconcepts shared by e_1 and e_2 are normalized by the union of the superconcepts.

The major shortcoming of computer-based similarity judgment is that the results do not fulfill all users' requirements (Janowicz 2008). This is mostly due to the lack of context information. Since the context is to be the crucial factor of the intelligent complex event processing, it also must be considered in the similarity calculation. Within the similarity calculation contextual information can influence the ranking list of the most similar events. In our approach the contextual information are considered in the feature-based similarity sim_{pr}.

$$sim_{pr}(e_1, e_2) := \frac{\omega_f * \frac{\alpha * v_{rcc} + \beta * v_{rc}}{\alpha * v_{rcc} + \beta * v_{rc} + \delta * v_{r-c} + \gamma * v_{c-r}}}{k}$$

The function f_{dist} calculates the similarity for every equivalent property in e_1 and e_2 under consideration of existing context information. The similarity is calculated by considering not only properties and the values but also the type of the property values. The calculation of similarity consists of elements which increase the similarity, $v_{rcc}$and v_{rc}, and elements which decrease the similarity, v_{r-c} and v_{c-r}. The context information is considered in v_{rcc}.

- v_{rcc} expresses how many values e_1 and e_2 have in common with respect to the context for each property;
- v_{rc} expresses how many values e_1 and Ie_2 have in common for each property;
- v_{r-c} expresses how many values are e_1 and not in e_2 for each property;
- v_{c-r} expresses how many values are e_1 and not in e_2 for each property.

The result of the calculation is normalized by the value k whereby k represents the number of distinct properties in

[5]Note that the role of condition could be added to the context part. However we want to emphasize the conceptual difference between explicit and implicit data, and hence importance of reasoning over situations.

both event instances. Like the weight-value used by the aggregation function we define a weight ω_f which expresses the importance of a property. The parts of f_{dist} which increase or decrease the similarity also have weights.

In the remaining part of this subsection we utilise the similarity measures defined above to define a set of operators for complex similar event specifications.

Let $\mathcal{E}$ denote a finite set of all known event types (i.e., all event classes defined in an event ontology). For each (atomic or complex) event type $E \in \mathcal{E}$, $DOM(E)$ represents the set of event instances of type E. For an event e, we say that it is an instance of type E iff $e \in DOM(E)$. We assume a discrete time model, where time $\mathcal{T}$ is an ordered set of time points t_0, t_1, t_2.... Time points are represented as integers, though other time models for time and data representation are possible without restrictions. An event e is defined over a time interval $\triangle t = [t_0, t_2] = \{t_1 | t_0 \leq t_1 \leq t_2\}, t_0, t_2 \in \mathcal{T}$. For convenience we define: $start([t_0, t_2]) = t_0$, $end([t_0, t_2]) = t_2$, and $[t_0, t_2] \cup [t_1, t_3] = [min\{t_0, t_1\}, max\{t_2, t_3\}]$. Finally an event is represented as a tuple $\langle E, e, \triangle t \rangle$, where E is an event type, e is an instance of that type, and $\triangle t$ is an interval over which the event happened[6].

In the following definition we formally introduce event calculus operators: the disjunction dis of E_0 and E_1 represents that either of E_0 or E_1 occurs, denoted $E_0 \vee E_1$. Conjunction con means that both events have occurred, possibly not simultaneously, and is denoted $E_0 + E_1$. The negation neg, denoted $E_0 - E_1$, occurs when there is an occurrence of E_0 during which there is no occurrence of E_1. A sequence seq (denoted $E_0; E_1$) is an occurrence of E_0 followed by an occurrence of E_1. Finally, there is a temporal restriction $E_0(\triangle t)$, which occurs when there is an occurrence of E_0 shorter than $\triangle t$ time interval, denoted tim.

Definition For similar events $\langle E_0, e_0, \triangle t_0 \rangle$ and $\langle E_0^{'}, e_0^{'}, \triangle t_0^{'} \rangle$ with the similarity measure $sim(e_0, e_0^{'})$, and similar events $\langle E_1, e_1, \triangle t_1 \rangle$, $\langle E_1^{'}, e_1^{'}, \triangle t_1^{'} \rangle$, with the similarity measure $sim(e_1, e_1^{'})$, define:

- **Disjunction:** $dis(E_0, E_1) = \{e_0 \vee e_1 | \triangle t = \triangle t_0 \vee \triangle t = \triangle t_1$, where e_0 may possible be replaced with $e_0^{'}$ and e_1 with $e_1^{'}$, in which case $\triangle t_0$ is replaced with $\triangle t_0^{'}$ and t_1 with $\triangle t_1^{'}$, respectively$\}$;
- **Conjunction:** $con(E_0, E_1) = \{e_0 \wedge e_1 | \triangle t = \triangle t_0 \cup \triangle t_1$, where e_0 may possible be replaced with $e_0^{'}$ and e_1 with $e_1^{'}$, in which case $\triangle t_0$ is replaced with $\triangle t_0^{'}$ and t_1 with $\triangle t_1^{'}$, respectively$\}$;
- **Negation:** $neg(E_0, E_1) = \{e_0 | \neg \exists e_1 (start(\triangle t_0) \leq start(\triangle t_0) \wedge end(\triangle t_1) \leq end(\triangle t_0))\}$;
- **Sequence,** $E_0; E_1$**:** $seq(E_0, E_1) = \{e_0 \wedge e_1 | end(\triangle t_0) < start(\triangle t_1)$, where e_0 may possible be replaced with $e_0^{'}$ and e_1 with $e_1^{'}$, in which case $\triangle t_0$ is replaced with $\triangle t_0^{'}$ and t_1 with $\triangle t_1^{'}$, respectively$\}$;
- **Temporal restriction:** $tim(E_0, \triangle t) = \{e_0 | end(\triangle t_0) - start(\triangle t_0) \leq \triangle t$, where e_0 may possible be replaced with $e_0^{'}$, in which case $\triangle t_0$ is replaced with $\triangle t_0^{'}$.

Using an operation defined in the previous definition we can create a new complex event and assign a name to it (e.g., a new type CE_0). Alternatively such an expression may repeatedly be used for building more complex events with other operations. Construction of complex event expressions requires also propagation of similarity measures upward the Situation Discrimination Tree (see Section Detection of Complex Events and Situations). For this purpose we consider every operation in the tree and calculate for each node the propagated value.

$$p(sim) := \sum_{depth=n}^{0} \sum_{nodeAtDepth=1}^{n} op(val_1, val_2)$$

The propagation function $p(sim)$ starts the calculation with the maximum depth level of the tree. Values are a combination of the calculated similarity and default values (which is 1), see also Figure 5. For each level it considers all nodes and invokes the calculation function op according to the node operator (see Definition 4.3 for the operators). For every operator, except the *negation* and *temporal restriction* operator, is a node similarity propagation defined:

$$or(val_1, val_2) = \begin{cases} val_1, & \text{iff } val_1 \geq val_2 \\ val_2, & \text{else} \end{cases}$$
$$and(val_1, val_2) = \frac{val_1 + val_2}{|val|}$$
$$seq(val_1, val_2) = and(val_1, val_2)$$

The calculation for the seq operator is the same as for the and operator since we do not consider any time constraints in the similarity calculation.

Context Model for Event Processing

Complex event processing (CEP) is about aggregation of atomic events in order to detect situations of particular interest. The events aggregation is specified in advance by event patterns, hence we say that CEP deals with *known* events. Extending the standard CEP with similarity search techniques (Section Event Calculus Extended With Similarity Search), our intention is also to tackle discovery of *unknown* events. Unknown events cannot be recognised with classical CEP approaches as they are not defined in advance. On the other side, unknown events may represent notable situations which do require reactions. An example is our use case, which deals with appropriate reactions on CRM server defects. Other scenarios typically include fraud and error detections etc.

Handling unknown events, and hence unknown situations, is a challenging task. We believe the context in which events are identified may be helpful in tackling this challenge. A context is a set of attributes and predetermined values, labeled in some meaningful way and associated with desirable semantics (Lassila and Khushraj 2005). Instead of a single value, attributes could be constrained over a range of values. We have created a context ontology which describes each context (of particular interest to our application) with corresponding actions relevant in that context. Furthermore

[6]For atomic events the time interval is equivalent to the time point.

the context is described with a number of attributes which differ for different contexts (e.g., location, execution phase, involved actors etc.). In order to dynamically bind the context attributes to specific instances in run-time, we execute a set of SWRL[7] rules (accompanied with the ontology[8]).

Sometimes triggering two or more actions by the same event may cause a conflict. Very often this is happening as those actions are supposed to be executed in different contexts. Hence introducing the context explicitly in reaction rules, and enabling reasoning over contexts help in the conflict resolution. Intuition behind this idea is to shift the issue of the conflict resolution to a problem of identifying conflicting situations (i.e., contexts). However in this direction of research we still do not have measurable results, and it is a topic of our future work.

Detection of Complex Events and Situations

Figure 3 shows an event discrimination tree from (Chakravarthy et al. 1994b). Leaves in the tree represent atomic events and nodes represent event calculus operators. The detection of complex events starts bottom-up as atomic events populate slots for leaves. When a condition (i.e., operation) specified in a particular node is satisfied with event instances[9], the recognised pattern is further propagated up to its node parent. Every node maintains its local memory of data (i.e., history of event occurrences). The history of events is important for implementation of different polices that prevent event processing from combinatorial explosion (i.e., detection of useful but also too many useless events). In (Chakravarthy et al. 1994b) four different policies have been defined, and for our use case we focus on the *recent* policy, i.e., only the last event occurrence needs to be remembered at each node.

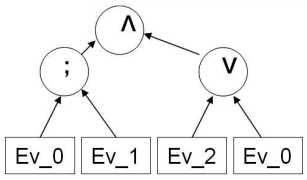

Figure 3: Event Discrimination Tree

In general it is very hard to control behavior of event-triggered reactive (EtR) systems. For instance, execution of an event may trigger other events, and these events may trigger even more events. There is neither guarantee that, such a chain of events will stop, nor that a reactive system executes actions as intended. We believe EtR systems would be more controllable if there was a tighter connection between events and actions. Standard ECA rules use the condition part (i.e., C) to establish that connection. Our approach is to use the context for that purpose (see Section Context Model for Event Processing). The context (Section Context Model for Event Processing) captures not just explicit background information, but also history of passed actions triggered by particular events. Therefore the context represents a relationship between events and actions. Due to this reason we assign to each arrow a context used for detection of a particular complex event, Figure 5. Unlike the event discrimination tree (Figure 3), the tree from Figure 5 represent a *situation* discrimination tree (as it detects events within certain contexts). Consider the rules from our use case in Section Motivating Use Case Scenario. We interpret a complex event *CRMMonitorEvent* in a context *InitialRepairContext* as situation s_0. Handling the situation s_0 requires an action *SuspendThreads*, which is different from handling the same event *CRMMonitorEvent* in a context *RepairContext*, i.e., situation s_1 (Figure 4). However as presented, in the situation s_1 another action (i.e., *SuspendThreads*) will be triggered.

ON *CRMMonitorEvent* WITHIN *InitialRepairContext* IF *Trashold* $> 90\%$ DO *SuspendThreads*.
ON *CRMMonitorEvent* WITHIN *RepairContext* IF *Trashold* $> 90\%$ DO *WarnMonitorCockpit*.

Figure 4: ECCA Sample Rules

The current run-time context is determined by a reasoner in each node of the tree (though the tree is constructed at the design time using complex event specifications and the context part of ECCA rules). Thus detection of a complex event depends both on atomic events and the context itself. Note that the tree will not even detect some complex events which are out of scope of an EtR system. Further on, the situation discrimination tree from Figure Situation Discrimination Tree is enriched with data to detect unknown events. What is an unknown event? We consider all atomic and complex events defined in an event ontology as known events (i.e., events that are fully specified). A complex event which do not fulfill a particular event pattern exactly, but to some extent, is an *unknown event*. Further on, as an event characterised with a certain context represent situation, an unknown event in a given context creates an *unknown situation*.

In Figure Situation Discrimination Tree, the relationship between two similar events is depicted with a dashed arrow. The semantics of a dashed arrow between two events is: *"replace a known (exact) event (in its absence) with a similar event"*. The intention here is not to trigger a complex event as soon as possible, possible replacing exact events with similar ones. If exact atomic events come (fulfilling specified time constraints) corresponding complex events will still be generated[10]. What we intend is just to provide a first step towards handling events that are not fully specified (i.e., unknown events). As it is difficult to process objects (i.e., events) in an automated and reasonable manner without knowing them in advance, we tackle this challenge by

[7]SWRL: www.w3.org/Submission/SWRL

[8]Due to space reasons we could not provide the content of rules in the paper. Instead, they are accessible in the ontology files from http://sake.fzi.de/

[9]Additionally it is ensured that such a set of instances has not already been detected.

[10]This is one of tasks of memory buffer allocated with every tree discrimination node.

trying to process those that are similar to known ones.

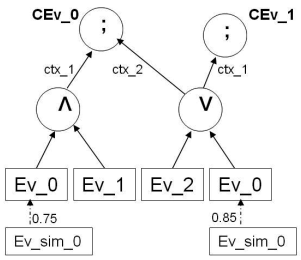

Figure 5: Situation Discrimination Tree

We express a level of similarity between two events by assigning the weight (i.e., similarity measure) to each dashed arrow. The value of weight is calculated and propagated upward based on similarity measures described in Section Event Calculus Extended With Similarity Search. Calculating similarities between events we are effectively establishing more relations between complex event patterns. As this process involve reasoning over an event ontology, computationally it may be very expensive. However note that this task can be done at the design phase. Also note that having both, an event ontology with events specification and an event discrimination tree, it seems we have defined events twice. That is not true as the role of an event ontology is to specify events in order to discover similarities of them and process events within particular contexts. On the other hand, the role of an event discrimination tree is to detect complex events and situations as soon as they occur (what is not possible with *static* ontologies).

Conclusion

In this paper we presented a novel approach for modeling event-triggered reactivity. Our event processing model with similarities measures is used in a distributed web environment to help in monitoring of the system execution. The approach extends the traditional event processing by introducing the concept of contextualized similarity, which enables the discovery of unknown complex events. Our initial evaluation has shown that the method is able to detect unknown events with a high relevance. The presented approach opens the possibility to define the reactivity of a system in more complete manner, and can be applied in various application domains in which such reactivity is of particular importance.

References

Adaikkalavan, R., and Chakravarthy, S. 2006. Snoopib: Interval-based event specification and detection for active databases. *Data Knowl. Eng.* 59(1):139–165.

Andreasen, T.; Bulskov, H.; and Knappe, R. 2003. From ontology over similarity to query evaluation pp. 39-50, in r. bernardi, m. moortgat (eds.): 2nd colognet-elsnet symposium - questions and answers: Theoretical and applied perspectives, amsterdam, holland.

Chakravarthy, S.; Krishnaprasad, V.; Anwar, E.; and Kim, S. K. 1994a. Composite events for active databases: Semantics, contexts and detection. In Bocca, J. B.; Jarke, M.; and Zaniolo, C., eds., *20th International Conference on Very Large Data Bases, September 12–15, 1994, Santiago, Chile proceedings*, 606–617. Los Altos, CA 94022, USA: Morgan Kaufmann Publishers.

Chakravarthy, S.; Krishnaprasad, V.; Anwar, E.; and Kim, S.-K. 1994b. Composite events for active databases: Semantics, contexts and detection. In *VLDB '94: Proceedings of the 20th International Conference on Very Large Data Bases*, 606–617. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.

Gehani, N. H.; Jagadish, H. V.; and Shmueli, O. 1992. Composite event specification in active databases: Model & implementation. In *VLDB '92: Proceedings of the 18th International Conference on Very Large Data Bases*, 327–338. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.

Hirakawa, H.; Xu, Z.; and Haase, K. 1996. Inherited feature-based similarity measure based on large semantic hierarchy and large text corpus.

Janowicz, K. 2008. Kinds of contexts and their impact on semantic similarity measurement. In *5th IEEE Workshop on Context Modeling and Reasoning (CoMoRea) at the 6th IEEE International Conference on Pervasive Computing and Communication (PerCom08)*.

Lassila, O., and Khushraj, D. 2005. Contextualised applications via semantic middleware. In *In Proc. of the Second Annual Conference on Mobile and Ubiquitous Systems (MobiQuitous '05)*.

Maedche, A.; Staab, S.; Stojanovic, N.; Studer, R.; and Sure, Y. Semantic portal - the seal approach. in: Spinning the semantic web, d. fensel, j. hendler, h. lieberman, w. wahlster (eds.),pages 317-359, mit press, 2003.

McCarthy, D., and Dayal, U. 1989. The architecture of an active database management system. In *SIGMOD '89: Proceedings of the 1989 ACM SIGMOD international conference on Management of data*, 215–224. New York, NY, USA: ACM.

Paton, N. W., and Díaz, O. 1989. Active database systems. In *ACM Comput. Surv.* ACM.

Pedersen, T.; Pakhomov, S. V. S.; Patwardhan, S.; and Chute, C. G. 2007. Measures of semantic similarity and relatedness in the biomedical domain. *J. of Biomedical Informatics* 40(3):288–299.

Stojanovic, N; Anicic, D. 2008. Towards creation of logical framework for event-driven information systems. In *To appear in: 10th International Conference on Enterprise Information Systems*.

Wu, Z., and Palmer, M. 1994. Verb semantics and lexical selection. In *32nd. Annual Meeting of the Association for Computational Linguistics*, 133 –138.

Zhang, X.; Jing, L.; Hu, X.; Ng, M.; and Zhou, X. 2007. A comparative study of ontology based term similarity measures on pubmed document clustering. In *DASFAA*, 115–126.