A Language and an Execution Model for the Detection of Active Situations

Asaf Adi[1] adi@il.ibm.com

Information Systems Engineering Area
Faculty of Industrial Engineering and Management
Technion - Israel Institute of Technology

IBM Research Laboratory in Haifa
ISRAEL

Abstract

This paper presents a thesis about a language and an execution model for the detection of situations aimed at reducing the complexity of active applications. This work has been motivated by the observation that in many cases, there is a gap between current tools that enable to react to a single event (following the ECA: Event – condition – action paradigm), and the reality, in which a single event may not require any reaction, however the reaction should be given to patterns over the event history. The concept of **situation** presented in this paper, extends the concept of **composite event**, in its expressive power, flexibility, and usability. This paper motivates the work, surveys other efforts in this are, and presents preliminary ideas for both the language and the execution model.

1. Introduction

In recent years, a substantial amount of work has been invested in systems that either react automatically to actual changes (reactive systems), or to predicted changes in their environment (proactive systems). These systems perform actions or signal alerts in response to the occurrence of events that are signaled when changes in the environment occur (or inferred). Such systems are used in a wide spectrum of areas and include command and control systems, active databases, system management tools, customer relationship management systems and e-commerce applications.

A central issue in reactive and proactive systems is the ability to bridge the gap between the events that are identified by the system and the **situations,** the cases to which the system is required to react. Some examples, from various areas, of situations that need to be handled are shown in figure 1.

There are a variety of tools that have been constructed to provide work environment for event driven applications. The work described in this paper has been motivated by the observation that most of the contemporary tools can react to the occurrence of a single event. In many applications (including all the examples shown in figure 1) the customer wishes to react to the occurrence of a *situation***,** which is a semantic concept in the customer's domain of discourse. The syntactic equivalent of a situation is a (possibly complex) pattern over the event history. Thus, there is a gap between applications' requirements and the capabilities of the supporting tools, resulting in excessive work. This thesis is aimed at developing methodology (a language and execution model) to bridge this

[1] Asaf is a Ph.D. candidate at the Technion – Israel Institute of Technology where he performs his research under the supervision of Dr. Opher Etzion. He is a research staff member in the Active Management Technology group at the IBM Research Laboratory in Haifa, Israel.

gap and save the resulted excessive work. It should be noted that the "pattern over the event history" may in some cases be only an approximation of the actual situation, or express the situation with some level of uncertainty. In this work we have made the simplified assumption of equivalence between these two terms.

- A client wishes to activate an automatic "buy or sell" program, if a security that is traded in two stock markets, has a difference of more than five percent between its values in the markets, such that the time difference between the reported values is less than five minutes ("arbitrage").
- A customer relationship manager wishes to receive an alert, if a customer's request was reassigned at least three times.
- A groupware user wishes to start a session when there are ten members of the group logged in to the groupware server.
- A network manager whishes to receive an alert, if the probability that the network will be overloaded in the next hour is high.

Figure 1 – Possible situations

Some tools and some research prototypes approach this difficulty by providing a mechanism for the definition of *composite events* that are detected when a predicate over the event history is satisfied. However, previous research was focused on specific fields such as active database [3][9][17] and network management [14][16], and resulted in partial solutions that have limited expressive power and can only be used in these specific domains by systems to which they were specially designed. Moreover, these prototypes are not able to fully express some of the basic elements of situation definition. These elements are described by the following requirements:

1. The events that can participate in situation detection.
2. The context during which situation detection is relevant.
3. The impact of the semantic information that is reported with events on situation detection (i.e. the semantic conditions that must be satisfied in order to detection a situation).
4. The decision possibilities about the reuse of event instances that participated in situation detection. The decision is whether, and on which conditions, the event instance is "consumed" and cannot be used for the detection of other situations.
5. The execution order when two or more events occur simultaneously or when an event has multiple roles in a situation.
6. The reflection of a correct order of events when there is a distinction between the time in which the event happens in reality and the time it is detected by the system. Phenomena of long delays in event reporting and events that are reported not according to their real order are ignored in current systems. Moreover, these systems assume equality of these two time points and base the composition upon the order in which events are detected.

This paper presents a thesis that addresses the situation concept, and defines a language and an execution model for the detection of active situations. In section 2, we review some previous work; in section 3, we outline the proposed solution, the methodology that is used to accomplish it, and the results achieved so far; and in section 4 we describe the thesis contribution.

2. Related Work

Contemporary **commercial systems** do not support composite events. However, they support triggers as specified in the SQL3 standard [11]. A trigger in SQL3 is an ECA rule that is activated by a database state transition and has an SQL3 predicate as a condition and a list of SQL3 statements as an action. Commercial databases that support triggers include DBMS products such as DB2, Oracle, Sybase and Informix.

Research on complex events for **active databases** is quite comprehensive and several research prototypes have been proposed, most of them are basing their event composition capabilities on some kind of event algebra.

1. ODE [9] is limited to database events only. It detects composite events over an event history that contains all event occurrences (i.e. ODE can not express time interval during which situation detection is relevant) and forbids the reuse of event instances (i.e. events are always consumed). Although semantic information is reported with events in ODE, this information can only be used to impose some filtering conditions (masks) and equality conditions (parameters) on events that participate in an event expression (composite event).
2. Snoop [3] supports both database event and external events (the semantics of external events are not described). It has limited expressive capabilities for the definition of time internals using the operators A, A*, P, and P* in association with a parameter context. Parameter contexts describe some decision possibilities for event reuse (consumption). However, Snoops cannot express all possibilities of event reuse (consumption) policies. Although semantic information is reported with events in Snoop, this information cannot be used during event composition (it can be used in the condition part of the ECA rule).
3. Zimmer’s and Unland’s model [17] support both database event and external events. It does not define the time interval during which situation detection is relevant and supports only few, predetermined, event reuse (consumption) policies. Semantic information is reported only with database event. This information can only be used to impose equality conditions on composing events.

Additional research prototypes of complex events for active databases including EXACT [6], REACH [18], ACOOD [2], ROCK & ROLL [7], Chimera [12], and REFLEX [13] do not offer new functionality. Other prototypes offer new functionality by introducing new operators. These include HiPAC [5], NAOS [4], and SAMOS [8] that deals with the detection of complex events using colored petri-nets in addition to the introduction of a new operator. Additional prototypes that are not based on event algebra, but on functional programming and real time logic include PFL [15] that is based on functional programming; JEM [10], that is based on the logic RTL (Real Time Logic); and ADL [1].

Event correlation (network management) systems (HP openView Event Correlation Services [14], SMARTS InCharge [16], VERITAS NerveCenter [19]) are designed to handle only network events. Their expressive power is limited to the network management domain and they do not aim at providing a

general (domain independent) solution that supports the fundamentals of a situation definition we described earlier.
We have shown that none of these prototypes and systems is a comprehensive solution that satisfies the requirements we have defined. We have shown that these solutions suffer from a deficiency in their expressional power, inaccuracies in their semantics and a centric approach that prevent the possibility to use these solutions in most of the real world applications. We have shown that a comprehensive solution to these problems is needed.

3. Proposed Solution

The research goal is to **define a language and an execution model for the detection of active situations** that satisfies all the requirements detailed in the introduction.

3.1. Methodology

The methodology consists of the following activities:

- Review several case studies of active systems (e.g. previous work, active applications) to identify requirements for a situation definition language.
- Define the situation language syntactically (XML) and formally (first order logic).
- Define an execution model (pseudo code algorithms) to enable situation detection.
- Build a prototype that demonstrates both the language and the execution model

The activities detailed above aim at identifying the gap between contemporary solution and the requirements, suggesting a language that bridge the identified gap, suggest an efficient execution model that implements the language, and demonstrated both the language and the execution model by building a prototype.

3.2. Results achieved so far

3.2.1. Case study – reactive distributed traffic control system

A traffic control system consists of control units that are placed in every junction. A control unit is responsible for the operation of traffic lights in a junction. It receives events from sensors that detect vehicles and pedestrians, traffic light buttons, other neighbor control units and external sources such as citizens reporting about traffic load. The control units detect situations in which the traffic lights should change, or the traffic load in the junction is too high.
The control unit that is responsible for the junction is described in figure 2. It receives events from nine sensors: four pedestrian sensors, three vehicle sensors and two traffic-light buttons for pedestrians. A pedestrian sensor reports an event to the control unit for every pedestrian that waits for the light to change. It also reports an event for every pedestrian that stops waiting. A vehicle sensor operates in the same method. Whenever a pedestrian presses a button on a traffic light, an event is reported to the control unit.

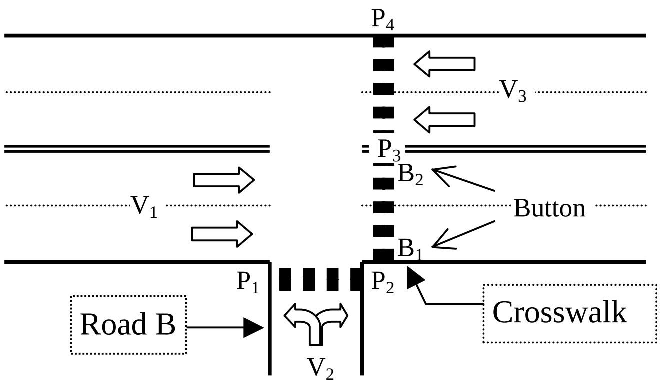

Figure 2 – A junction

3.2.2. Event model

An event is a significant (in some domains) instantaneous (happens in a specific point in time), atomic (happens completely or not at all) occurrence that is triggered by a transition between states. It is materialized by an *event instance* that contains the necessary information about the event.

An *event class* describes the common properties of a similar set of event instances on an abstract level. It defines the type of information that can be associated with the event through event attributes.

The event model defines the event classes and the their semantic relationships with other events and entities (e.g. classification of an event instance to an event class). The event *buttonPressed* occurs whenever a pedestrian button is pressed. This event has three attributes: *buttonId* of type string, *junctionId* of type number and *pressingTime* of type chronon (indicates a timestamp).

(B_1, 1, 08:00:00) is an instance of the event *buttonPressed*. *ButtonId*, which is the first attribute, has the value B_1; *JunctionId*, which is the second attribute, has the value *1*; and *pressingTime*, which is the last attribute, has the value *08:00:00*. This event instance occurred at eight o'clock when button B_1 in junction *1* was pressed.

The event *buttonPressed* is a specialization of the event *junctionEvent* that is triggered whenever a significant event occurs in the junction.

Figure 3 – Event model

3.2.3. Event group

Event group is a collection of semantically associated event instances (e.g. events that reports about the same junction belong to the same event group). Event groups are used to match different event instances that refer to the same entity or concept. An event group divides the situation detection process to numerous separate independent detection process (denote partitions); one partition for every event group.

An *event group class* defines a set of *grouping expressions* that semantically associate event instances with an event group or several event groups according to specific semantics (examples are: all pedestrians that wait for the same cross light, all vehicles that are on the same road). The result of a grouping expression over the information that is associated with an event, denoted *group value*, describes the event group.

Three event groups partition situation that detects vehicle overload in a group of junctions that are in the same area. These event groups associated events that report about vehicles at the northern part of the city (junctions 1 to 5), the southern part of the city (junction 6 to 10) and the city center (junctions 11 to 15).
The same situation can be used to detect vehicle overload in a single junction if event groups that associates events that report about vehicles in a single junction partition it.

Figure 4 – Event group

3.2.4. Lifespan

A *lifespan* is a temporal interval during which situation detection is relevant. It is bounded by two events called *initiator* and *terminator*. An occurrence of an initiator event initiates the lifespan and an occurrence of a terminator event terminates it.
A *lifespan class* describes the common properties of a similar set of lifespans on an abstract level. It defines the set of events that can initiate a lifespan, the set of events that can terminate it, the conditions for the lifespan's initiation and termination, and the lifespan's maximal length.

The situations *buttonPressedThreeTimes* and *lightIsRedFiveMinutes* are relevant, and detected, during a lifespan that is initiated when the light in crosswalk *A* becomes red and is terminated when the light turns green again.
The situation *congestionInRoadB* is relevant, and detected, during a lifespan that starts every three minutes and is terminated five minutes after it starts thus more than one lifespan of this class exist simultaneously.

Figure 5 – Lifespan

3.2.5. Context

A *context* is a semantic notion that describes partial knowledge about the world's state. It is a combination of one or more partitioning expressions and a predicate. A partitioning expression explicitly defines a set of states that has common semantics. Examples are temporal element, area, and semantic connotation. In this work, we define a context as a combination of a temporal element (lifespan), a semantic connotation (event group) and a predicate. The temporal element designates the time in which the context is active if the predicate evaluates to true. The semantic connotation designates event instances that are relevant in the context. Note that a single context is associated with (opened for) every combination of an event group and a lifespan (one to one relationship).
One or more situations can be associated with a context. A situation that is associated with a context is detected while the context is active. The decision, whether a situation has occurred, considers only event instances that occur while the context is active and are relevant in it.

The situations *buttonPressedThreeTimes* and *lightIsRedFiveMinutes* are relevant, and detected, in a context that is consist of

1. A lifespan that is initiated when the light in crosswalk *A* becomes red and is terminated when the light turns green again.
2. An event group that associates event instances originated from the same junction
3. A predicate that checks that an operator did not override the control unit.

Figure 6 – Context

3.2.6. Event collection

An *event collection* designates the event instances that are considered for situation detection, if they occur while the context that is associated with the situation is active. These event instances, denoted *candidates*, are partitioned into *candidate lists*. A candidate list is a collection of event instances (candidates) that have the same role in situation detection (e.g. in the situation *congestionInRodeB,* events that report about vehicles has one role, and events that report about pedestrians has other role). An event collection defines these candidate lists: conditions that events must satisfy in order to participate in a candidate list, condition for removing event instances from a candidate list, and decision possibilities about the reuse of event instances that participate in situation detection.

The situation *trafficJam* is detected if the total number of vehicles in the junction as detected by the sensors V_1, V_2, and V_3 is over 50 or the number of vehicles detected by a single sensor is over 30. An event collection for this situation consists of three candidate lists, one for every sensor. A candidate list holds events that report on vehicles that wait in the junction. The event collection defines that:

- Events detected by sensor V_i belong to the ith candidate list (conditions that events must satisfy in order to participate in a candidate list).
- Events are removed from a candidate list if the vehicle on which they reported, stops waiting in the junction (condition for removing event instances from a candidate list).
- Events that triggered situation detection continue to participate in situation detection (decision possibilities about the reuse of event instances that participate in situation detection).

Figure 7 – Event collection

3.2.7. Situation

A *situation* definition is an event algebra expression over a combination of a context and event collection. The context determines the lifespan during which the situation is detected and the situation’s semantic connotation (event group). The event collection determines the event instances that are considered for situation detection.

A situation consists of a combination of a *situation expression* and a *triggering expression*. The situation expression determines the conditions for situation detection, and the event instances that caused it. The triggering expression defines the event (or events) that is triggered as result of situation detection and its associated information.

The situation expression itself consists of a combination of an *operator* and *qualifiers* (designate a selection strategy when several candidates of an event that is in the domain of a situation operator exists), a *predicate* (applicable for certain operators), and *detection mode*. The combination of an operator and qualifiers designates an event pattern; the predicate designates a condition over the events in the pattern results in tuples of event instances that could have cause the situation; and the detection mode determines if a situation can be detected during the context (*immediate*) or at the end of it (*deferred*)

The language supports *joining operators* (*conjunction*, *disjunction*, *sequence*, *strictSequence*, *simultaneous*, and *aggregation*), s*election operators* (*first*, *until*, *since*, and *range*), *assertion operators* (*never*, *sometimes*, *last*, *min*, *max*, and *unless*), and *temporal operators* (*at*, *after*, and *every*).

- The situation *buttonPressedThreeTimes* is detected in the immediate detection mode by an aggregation operator.
- The situation *lightIsRedFiveMinutes* is detected in the immediate detection mode by a temporal operator.

Figure 8 – Situation

3.2.8. Prioritization algorithms

Prioritization algorithms identify cases in which the order of situation detection is undetermined and defines a mechanism for the definition of a deterministic detection execution. These cases take place when two events occur simultaneously or an event has multiple roles in situation.

In a junction with 49 vehicles is unclear if the situation *trafficJam* should be detected when two events, one that reports on a new vehicle that waits in the junction and another that reports on a vehicle that left the junction, occur simultaneously. Prioritization algorithms should identify such cases and suggest a solution strategy.

Figure 9 – Prioritization

3.2.9. Synchronization algorithms

Synchronization algorithms designate how to distinct between the time in which an event happens in realty and the time it is detected by the system and to overcome the phenomena that result from the gap between these times. Composition of situations, without applying some methods for synchronization, may result in the detection of situations that have not occurred in reality or in missing the detection of situations that occurred in reality.

The situation *trafficJam* can be detected by mistake, if there are 49 vehicles in the junction, and an event that reports on a new vehicle that in the junction arrives to the systems before an event that reports on a vehicle that left the junction, although these event occurs in the opposite order in reality.

Figure 10 – Synchronization

4. Contribution

The proposed output of this thesis is a powerful situation detection language and execution model that supports situations than cannot be defined by contemporary models. The main advantages of this work in comparison with existing models are:

- A comprehensive event model.
- Context management enables the detection of the same situation in parallel contexts that is determined by a combination of a temporal element (lifespan), a semantic connotation (event group) and a predicate.
- Powerful event algebra combined with partitioning, filtering, event expiration polices, and conditions results in an extensive expressional power of a situation language. The combination of event algebra operators and other elements of the langue extends the concept of composite event (that is based only on event algebra operators) in its expressive power, flexibility and usability.
- Deterministic order of execution when two or more events occur simultaneously or when an event has multiple roles in a situation.
- Situation detection is performed with respect to the event occurrence time and not the time in which the event is detect by the system.

5. References

[1] Behrends-H. "Simulation-based Debugging of Active Databases." Proceedings of IEEE International Workshop on Research Issues in Data Engineering: Active Databases Systems. Feb. 1994; Houston, TX, USA. IEEE Comput. Soc. Press, 1994. 172-180.

[2] Berndtsson-M. "ACOOD: an Approach to an Active Object Oriented DBMS" Master's thesis, Department of Computer Science, University of Skovde, Sweden. 1991.

[3] Chakravarthy-S, and Mishra-D. "Snoop: an expressive event specification language for active databases." Data and Knowledge Engineering 14.1 (1994): 1-26.

[4] Collet-C, and Coupaye-T. "Composite events in NAOS." Proceedings of the 7th International Conference on Database and Expert Systems Applications, DEXA. Sept. 1996; Zurich, Switzerland. Springer Verlag, 1996. 244-253.

[5] Dayal-U, Buchmann-A, and Chakravarthy-U. "The HiPAC Project." Active Database Systems: Triggers and Rules For Advanced Database Processing Morgan Kaufmann, 1996. 177-206.

[6] Diaz-O, and Jaime-A. "EXACT: an extensible approach to active object-oriented databases." VLDB Journal. 6.4 (1997): 282-295.

[7] Dinn-A, Paton-NW, Williams-MH, and Fernandes-AAA. "An Active Rule Language for ROCK & ROLL." Proceedings of the 14th British National Conferenc on Databases. July 1996; Edinburgh, UK. Springer Verlag, 1996. 36-55.

[8] Gatziu-S, and Dittrich-KR. "Events in an active object-oriented database system." proceedings of the 1st International Workshop on Rules in Database Systems. Sept. 1993; Edinburgh, UK Springer Verlag, 1994. 23-29.

[9] Gehani-NH, Jagadish-HV, and Shmueli-O. "Composite event specification in active databases: model and implementation." Proceedings of 18th International Conference on Very Large Data Bases. Aug. 1992; Vancouver, BC, Canada. Morgan Kaufmann, 1992. 23-27.

[10] Guangtian-Liu, Mok-AK, and Konana-P. "A unified approach for specifying timing constraints and composite events in active real-time database systems." Proceedings of 4th IEEE Real-Time Technology and Applications Symposium. 1998; Denver, CO, USA. IEEE Comput. Soc. Press, 1998. 199-208.

[11] Kulkarni-K, Mattos-NM, and Cochrane-R. "Active Database Features in SQL3." Active Rules in Database Systems. Springer Verlag, 1999. 197-219.

[12] Meo-R, Psaila-G, and Ceri-S. "Composite Events in Chimera." Proceedings of 5th Conference on Extended Database Technology (EDBT`96). March 1996; Avignon, France. Springer Verlag, 1996. 56-78.

[13] Naqvi-W, and Ibrahim-MT. "EECA: An Active Knowledge Model." Proceedings of 5th International Conference on Database and Expert Systems Applications. Sept. 1994; Athens, Greece. Springer Verlag, 1994. 380-389.

[14] Sheers-KR. "HP OpenView event correlation services." Hewlett Packard Journal. 47.5 (1996): 31-42.

[15] Swaup-R, Alexandra-P, and Carol-S. "PFL: An Active Functional DBPL." Active Rules in Database Systems. Springer Verlag, 1999. 297-308.

[16] Yemini-SA, Kliger-S, Mozes-E, Yemini-Y, and Ohsie-D. "High speed and robust event correlation." IEEE Communications Magazine. 34.5 (1996): 82-90.

[17] Zimmer-D, and Unland-R. "A General Model for Specification of the Semantics of Complex Events in Active Database Management Systems." C-LAB Report. 1998.

[18] Zimmermann-J, and Buchmann-A. "REACH." Active Rules in Database Systems. Springer Verlag, 1999. 263-277.

[19] "VERITAS NerveCenter^{tm}" VERITAS Software. http://eval.veritas.com/webfiles/docs/ NCOverview.pdf