Thesis Proposal

A resource-efficient event detection algebra

Jan Carlson
Department of Computer Science and Engineering
Mälardalen University, Sweden
jan.carlson@mdh.se

Abstract

Event detection is an important aspect of many application types, ranging from active databases over digital libraries and stock market agents, to reactive embedded systems. To allow systems to react to complex events patterns rather than to simple primitive events, an event algebra can be used.

We intend to develop an event algebra suitable for resource-conscious applications such as real-time and embedded systems. The algebra should have well-defined formal semantics, carefully designed to allow implementation with limited resources while at the same time retaining the intuitive properties of the operators.

1 Background and motivation

A wide range of applications, including active databases, monitoring systems and rule based embedded systems, are *reactive* in nature, meaning that the execution is driven by external events to which the system should react with an appropriate response. Events can be simple, e.g., sampled directly from the environment or occurring within the system, but it is often necessary to react to more sophisticated situations involving a number of simpler events that occur in accordance with some pattern.

A systematic approach to handle this type of systems is to separate the mechanism for detecting composite events from the rest of the application logic. The detection mechanism takes as input primitive events and detects occurrences of composite events which are used as input to the application logic. This separation of concerns facilitates design and analysis of reactive systems, as detection of complex events can be given a formal semantics independent from the application in which it is used, and the remaining application logic is free from auxiliary rules and information about partially completed patterns.

The mechanism to detect composite events can be constructed as an event algebra, i.e., a number of operators from which expressions can be built that represent the event patterns of interest. In order to allow formal reasoning about the behaviour of the system, it is essential that the algebra has a well-defined semantics. This is particularly important when the algebra is used in safety-critical applications for which formal verification is required. In addition, reasoning on a high level of abstraction is facilitated if the designer is provided a number of formal properties that the algebra conforms to. Such properties include, for example, laws of associativity and distributivity.

For embedded applications and systems with strict timeliness requirements, it is essential that the event detection can be implemented with limited resources. For a given complex event, defined in the algebra, one would like to know the maximum memory usage, or at least a safe approximation thereof, as well as the worst case execution time of the detection mechanism for that event.

2 Event algebra essentials

The following operations, or variants of them, are found in many event algebras. The *disjunction* of A and B represents that either of A and B occurs, here denoted $A \vee B$. *Conjunction* means that both events have occurred, possibly not simultaneously, and is denoted $A + B$. The *negation*, denoted $A - B$, occurs when there is an occurrence of A during which there is no occurrence of B. Finally, a *sequence* of A and B is an occurrence of A followed by an occurrence of B, and is denoted $A;B$. Additionally, we plan to include a *temporal restriction* construct. For example, $(A;B)_\tau$ occurs when an occurrence of A is followed by an occurrence of B within τ time units.

The operator semantics described informally above does not specify how to handle situations where an occurrence could participate in several occurrences of a composite event. For example, three occurences of A followed by two occurrences of B result in six occurrences of A+B. While this may be acceptable, or even desirable, in some applications, the memory requirements (each occurrence of A and B must be remembered forever) and the increasing number of simultaneous events means that it is unsuitable in many cases.

A common way to deal with this is to define the algebra in two steps. The operations are given an unrestricted, straightforward meaning. Then a restriction policy is applied to the operator, that acts like a filter so that only a subset of the occurrences allowed by the unrestricted definition are detected.

In most algebras, each complex event, including those that require more than one occurrence of simpler events in order to occur, is associated with a single time point (the time of detection, i.e., the time of the last occurrence that was required). Galton and Augusto [11] showed that this results in unintended semantics for some operation compositions. They suggest solving the problem by associating the occurrence of a complex event with the occurrence interval, i.e., the interval in which all required simpler events occurred, rather than the time of detection.

3 Related work

A lot of work, especially formal approaches, on event algebras has been done in the context of active databases. In addition, work in knowledge representation and general event notification services is also of relevance.

3.1 Active databases

One area where event algebras are used is active databases which, unlike passive databases, react automatically to situations that arise within or outside the database. The reactions are specified by so called *event-condition-action rules* (ECA rules) stating that when a certain event occurs, and the condition is satisfied, the given action should be performed. The event part of a ECA rule can be expressed by an event algebra to allow the database to react to complex events.

The event expression language uses in the object database Ode has the same expressive power as regular expressions, which allows the detection mechanism to be implemented by finite state automata [14]. The definition is based on a global, totally ordered set of primitive event occurrences, implying that primitive events can not occur simultaneously. To allow event occurrences to carry values and composite events that occur only under given restrictions on the values of the constituent events, the automata mechanism is extended with data structures that store the values of events that have occurred.

In the active database SAMOS, event detection is implemented using Petri nets [12, 13]. Event occurrences are associated with a number of parameter-value pairs, and it can be specified that a complex event should occur only if the constituent event occurrences have

the same value for a given parameter. SAMOS does not allow simultaneous primitive event occurrences.

Snoop [10, 9] is an event specification language for active databases. It defines four different restriction policies (called parameter contexts) that can be applied to the operators of the algebra. The unrestricted context is defined formally, but for the restriction policies only informal descriptions are given. The detection mechanism is based on trees corresponding to the event expressions, where primitive event occurrences are inserted at the leaves and propagate upwards in the tree as they cause more complex events to occur.

None of the algebras described above provide algebraic properties for their respective operators, and little is said about the memory and time complexity associated with the detection of complex events.

A formalized schema for this type of event detection, including a definition of the operations and restriction policies of Snoop using this schema, has been defined by Mellin and Andler [22]. The operators have definitions parameterised on restriction policies, which facilitates formal reasoning about the operators with different restriction policies applied, without requiring explicit definitions for each operator- restriction combination. They propose, as future work, to extend the operators with temporal constraints, which would allow an investigation of the temporal complexity of the detection algorithm.

Zimmer and Unland present a formal restriction framework in which the event algebras of Snoop, SAMOS, Ode and a few other systems are compared [28]. They also highlight a number of ambiguities and inconsistencies of the various approaches.

Liu et al. uses Real Time Logic to define a system where composite events are expressed as timing constraints and handled by general timing constraint monitoring techniques [21]. They present a mechanism for early detection of timing constraint violation, and show that it is possible to calculate a upper bound on the length of the structures needed to detect an event. In general, the time complexity of detection is in $O(n^3)$, but for a certain subset of expressions, a $O(n)$ algorithm is possible [23, 24].

The algebra presented by Baily and Mikulás [3], including four restriction policies, is defined formally in temporal logic. They identify a class of complex events for which testing whether two complex events are equivalent is decidable, and show that testing for implication is undecidable.

3.2 Event notification systems

Many systems and frameworks have been developed where clients register their interest in certain types of events with a central server. The server monitors the environment and, upon the detection of an event, notifies the concerned clients. As the clients typically perform monitoring tasks, they are generally interested in particular event sequences rather than single occurrences. Rather that having each application implement this separately, it is beneficial to extend the server to allow registration of complex event descriptions. The system presented by Hayton et al. [16] contains a simple event algebra, implemented using a pushdown automata.

The event algebra developed by Hinze and Voisard is designed to suit event notification service systems in general [18, 17]. Their algebra contains time restricted sequence and conjunction, which permits events like *A occurs less than t time units before B* to be expressed. Following the framework presented by Zimmer and Unland [28], the algebra is parameterised with respect to policies for event instance selection and consumption.

Additional examples of event notification systems that allow composite events expressed by an event algebra are the READY system [15] and the event specification language developed by Zang and Unger [27].

3.3 Knowledge representation

In the area of knowledge representation, similar techniques are used to reason about event occurrences. Interval Calculus introduce formalised concepts for properties, actions and events, where events are expressed in terms of conditions for their occurrence [1], and Event Calculus [20, 19] also deals with the occurrences of events. However, the motivation is slightly different from that of event algebras. Rather than detecting complex events as they occur, the focus of Event Calculus is to express formally the fact that some event has occurred, and to allow inferences to be made from it.

In the area of temporal data mining, event operators similar to those in event detection are used. A crucial difference is that the task of event detection is to detect patterns that match a given event description, while in data mining the data is analysed in search for trends and patterns for which matching descriptions are to be derived [2].

4 Research description

We intent to develop an event algebra suitable for applications where the ability to reason formally about the system is required, and where resources such as memory and time are limited. The algebra should be implemented together with an existing language for reactive systems.

4.1 Detailed description

In order to allow formal reasoning about the algebra or a system that uses it, we believe that a fully formal definition of the algebra is essential. We will use techniques such as interval-based semantics and restriction policies to handle complexity issues while retaining an intuitive semantics for the operators.

A key factor is the development of a suitable restriction policy. It should be restrictive enough to permit the formulation of memory consumption bounds, while at the same time allowing a simple formal relation between the restricted semantics and the unrestricted version. Such a relation facilitates formal reasoning since it allows, for example, properties to be proved for the simple, unrestricted semantics and then applied to the restricted version by means of this relation. These two objectives are contradictory to some extent, and careful investigation is required to find an optimal balance.

An implementation of the algebra is required to investigate time and space complexity in more detail. Preferably, the implementation is designed to work together with an existing language for reactive systems. This would give, in addition to an evaluation of the algebra itself, feedback on to how well the proposed algebra suits this domain.

Preliminary results indicate that as a result of ensuring some of the desired properties, only a subset of all expressions will be possible to detect with constant memory. In this case, it is important to clearly identify this class of expressions. Also, in some cases it might be possible to apply the algebraic properties to transform an expression into a form which belongs to the class of well-behaved expressions.

4.2 Project plan

The algebra The semantics of the event algebra, including the restriction policy, should be defined. Our restriction policy must be carefully designed to retain many of the operator properties that are intuitive. These include, for example, associativity of sequence, conjunction and disjunction; commutativity of conjunction and disjunction; and distributative laws. Additionally, we will investigate the relation between the restricted and the unrestricted semantics.

Operational semantics An imperative algorithm will be developed, and we will prove that it is equivalent to the declarative semantics of the algebra. This algorithm should be analysed with respect to resource requirements, and criteria under which it can be guaranteed to execute with bounded resources should be identified.

Transformations We will investigate the possibility of developing semantic-preserving expression transformations in order to increase the class of expressions that can be detected with bounded resources. Preferably, a transformation algorithm should be developed, as well as a precise definition of the class of expressions that, after applying the algorithm, can be detected with bounded resources.

Implementation The algebra will be implemented to work in concert with some existing language for reactive systems. Candidate languages include the functional reactive language suite AFRP based on time varying behaviours and discrete events [26, 25], the object-oriented reactive language Timber [8], and the synchronous language Esterel [4, 5].

Evaluation and experiments Once an implementation is developed, case studies can be carried out in order to evaluate the practical relevance of the algebra. This would also provide the basis for a discussion on how to extend the algebra, for example in terms of additional operators or restriction policies.

4.3 Preliminary results

A first version of the algebra was presented in [6], including the declarative and operational semantics, a number of algebraic properties and a short discussion regarding the resource weakness.

This algebra was extended with a temporally restricted sequence operator in [7]. This article also contains correctness proofs for the algorithm, additional properites of the operators, and a description of a class of expressions that can be detected by an implementation with a constant memory bound.

Currently, a revised version of the declarative semantics is being developed which uses a simpler restriction policy. As a result, the relation between the restricted and the unrestricted version is strengthened, at the cost of a worse average-case memory and time usage.

4.4 Timeplan

This section gives an overview of the planned activities.

2000 - 2003
Courses (37 credits)
Literature study
Developing the algebra

Spring 2004
Finishing a first complete version of the algebra (declarative and operational semantics, resource analysis and expression transformations) for the licentiate thesis.

Fall 2004
Course in Database Systems
Implementing the algebra

Spring 2005
Courses, including Distributed Systems and Software Engineering
Evaluation and experiments

Fall 2005
Courses
Developing the final version of the algebra

Spring 2006
Finishing the final version of the algebra for the PhD thesis

4.5 Milestones

The following milestones are defined for the project:

State-of-the-art report	Aug	2002
Publication FORMATS03	Sep	2003
Licentiate proposal (MdH)	Nov	2003
Thesis proposal (CUGS)	Feb	2004
Licentiate thesis	Apr	2004
PhD proposal (MdH)	Jun	2005
PhD thesis	Jun	2006

References

[1] J. F. Allen and G. Ferguson. Actions and events in interval temporal logic. *Journal of Logic and Computation*, 4(5):531–579, October 1994.

[2] C. M. Antunes and A. L. Oliveira. Temporal data mining: An overview. In *KDD Workshop on Temporal Data Mining*, pages 1–13, San Francisco, CA, 26 August 2001.

[3] J. Bailey and S. Mikulás. Expressiveness issues and decision problems for active database event queries. In *Database Theory - ICDT 2001, 8th International Conference*, volume 1973 of *Lecture Notes in Computer Science*, pages 68–82, London, UK, 4–6 January 2001. Springer.

[4] G. Berry. *The Esterel-V5 Language Primer*. CMA and Inria, Sophia-Antipolis, France, v 5.21, release 2.0 edition, May 1999.

[5] G. Berry. The foundations of esterel. In G. Plotkin, C. Stirling, and M. Tofte, editors, *Proof, Language, and Interaction: Essays in Honour of Robin Milner*, pages 425–454. MIT Press, 2000.

[6] J. Carlson and B. Lisper. An interval-based algebra for restricted event detection. In *Proceedings of First International Workshop on Formal Modeling and Analysis of Timed Systems (FORMATS 2003)*, Marseille, France, 6–7 September 2003.

[7] J. Carlson and B. Lisper. An event detection algebra for reactive systems, 2004. Submitted.

[8] M. Carlsson, J. Nordlander, and D. Kieburtz. The semantic layers of Timber. In *Proceedings of the First Asian Symposium on Programming Languages and Systems (APLAS'2003)*, Lecture Notes in Computer Science, Beijing, China, 26–29 November 2003. Springer-Verlag. To appear.

[9] S. Chakravarthy, V. Krishnaprasad, E. Anwar, and S.-K. Kim. Composite events for active databases: Semantics, contexts and detection. In *20th International Conference on Very Large Data Bases*, pages 606–617, Santiago, Chile, 12–15 September 1994. Morgan Kaufmann Publishers.

[10] S. Chakravarthy and D. Mishra. Snoop: An expressive event specification language for active databases. *Data Knowledge Engineering*, 14(1):1–26, 1994.

[11] A. Galton and J. C. Augusto. Two approaches to event definition. In *Proc. of Database and Expert Systems Applications 13th International Conference (DEXA'02)*, volume 2453 of *Lecture Notes in Computer Science*, pages 547–556, Aix-en-Provence, France, 2–6 September 2002. Springer-Verlag.

[12] S. Gatziu and K. R. Dittrich. Events in an active object-oriented database system. In *Proc. 1st Intl. Workshop on Rules in Database Systems (RIDS)*, Edinburgh, UK, September 1993. Springer-Verlag.

[13] S. Gatziu and K. R. Dittrich. Detecting composite events in active database systems using petri nets. In *Research Issues in Data Engineering (RIDE '94)*, pages 2–9, Los Alamitos, Ca., USA, February 1994. IEEE Computer Society Press.

[14] N. Gehani, H. V. Jagadish, and O. Shmueli. COMPOSE: A system for composite specification and detection. In *Advanced Database Systems*, volume 759 of *Lecture Notes in Computer Science*. Springer, 1993.

[15] R.E. Gruber, B. Krishnamurthy, and E. Panagos. The architecture of the READY event notification service. In *Proceedings of the 19th IEEE International Conference on Distributed Computing Systems, Middleware Workshop*, Austin, TX, USA, May 1999.

[16] R. Hayton, J. Bacon, J. Bates, and K. Moody. Using events to build large scale distributed applications. In *Proceedings of the ACM SIGOPS European Workshop*, 1996.

[17] A. Hinze and A. Voisard. Composite events in notification services with application to logistics support. Technical Report tr-B-02-10, Freie Universitaet Berlin, 2002.

[18] A. Hinze and A. Voisard. A parameterized algebra for event notification services. In *Proc. of the 9th Int. Symposium on Temporal Representation and Reasoning (TIME 2002)*, Manchester, UK, July 2002. Springer-Verlag.

[19] R. Kowalski. Database updates in the event calculus. *The Journal of Logic Programming*, 12:121, January 1992.

[20] R. A. Kowalski and M. J. Sergot. A logic-based calculus of events. *New Generation Computing*, 4:67–95, 1986.

[21] G. Liu, A. Mok, and P. Konana. A unified approach for specifying timing constraints and composite events in active real-time database systems. In *4th IEEE Real-Time Technology and Applications Symposium (RTAS '98)*, pages 199–209, Washington - Brussels - Tokyo, June 1998. IEEE.

[22] J. Mellin and S. F. Adler. A formalized schema for event composition. In *Proc. 8th Int. Conf on Real-Time Computing Systems and Applications (RTCSA 2002)*, pages 201–210, Tokyo, Japan, 18–20 March 2002.

[23] A. Mok and G. Liu. Early detection of timing constraint violation at runtime. In *The 18th IEEE Real-Time Systems Symposium (RTSS '97)*, pages 176–186, Washington - Brussels - Tokyo, December 1997. IEEE.

[24] A. Mok and G. Liu. Efficient run-time monitoring of timing constraints. In *Proceedings of the Third IEEE Real-Time Technology and Applications Symposium (RTAS '97)*, pages 252–262, Washington - Brussels - Tokyo, June 1997. IEEE.

[25] H. Nilsson, A. Courtney, and J. Peterson. Functional reactive programming, continued. In *Proceedings of the 2002 ACM SIGPLAN Haskell Workshop (HASKELL-02)*, pages 51–64, New York, October 3 2002. ACM Press.

[26] Z. Wan and P. Hudak. Functional reactive programming from first principles. *ACM SIGPLAN Notices*, 35(5):242–252, May 2000.

[27] R. Zhang and E. Unger. Event specification and detection. Technical Report TR CS-96-8, Department of Computing and Information Sciences, Kansas State University, June 1996.

[28] D. Zimmer and R. Unland. On the semantics of complex events in active database management systems. In *Proceedings of the 15th International Conference on Data Engineering*, pages 392–399. IEEE Computer Society Press, 1999.