A Constant-memory Event Algebra with Intuitive Algebraic Properties[*]

PhD Thesis Proposal

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

MÄLARDALEN UNIVERSITY

[*]This work was in part funded by the Swedish National Graduate School in Computer Science (CUGS).

1 Background and Related Work

Events exist in many different forms in different areas of software design and implementation. On lower abstraction levels, interrupt events represent that something has happened in the environment that the system might want to react to, for example a key being pressed or a sensor update. Real-time systems are typically designed to react either to events generated by external sensors, or by periodically occurring timer events [22].

In some programming languages, like Visual Basic and JavaScript, the overall program execution is driven by events. Other programming languages support event-based communication and control transfer as a complement to the ordinary language mechanisms, either as an integrated part of the language, e.g., in Java and C#, or through libraries or extensions. In particular, an event-driven programming style is often used for graphical user interfaces.

On a somewhat higher level, whole systems can be designed according to an event-based architectural style, meaning that the communication between different parts of the system is based on a publish/subscribe interaction paradigm [15]. Event consumers express an interest in certain events by registering a subscription with an intermediary event manager. When an event is published, it is matched against the current subscriptions and relayed to the appropriate consumers. The key features of the publish/subscribe paradigm include spatial and temporal decoupling, since the publisher and subscriber need not be aware of each other and can communicate asynchronously without blocking.

Middleware, i.e., software located between the operating system and the applications [24], facilitates the design of complex, distributed systems by hiding low-level details related to distribution and the underlying operating system and hardware. Event-based middleware, e.g., Hermes [28] and READY [20], provide a uniform high-level interface of event related services, which allows seamless event handling also between heterogeneous subsystems.

On an even higher level, event handling is useful when managing, monitoring or exploring complex systems, including large software systems or networks but also real-world systems like stock markets or news report services. Here, a main concern is dealing effectively with very large volumes of event occurrences, and to filter out only those that are of interest in a particular situation. Examples of work in this category include monitoring of real-time systems [27], supervision of telecommunication networks [14] and air traffic control [26].

1.1 Event Patterns and Event Algebras

In many applications, individual event occurrences are not the main point of concern, but rather the occurrences of certain event patterns. To support this, an event framework can provide means to specify event patterns, and allow these patterns to be used in the same way as ordinary events. Thus, the details of pattern detection is moved from the application to the event framework. For example, a subsystem might subscribe to the pattern *"A and B occurs within 2 seconds"*, instead of subscribing to A and B and perform the detection of the desired situation internally.

A number of methods have been proposed to specify event patterns in different settings. For example, some frameworks use regular expressions or finite automata. A more general approach is to use some variant of modal or temporal logic, possibly with explicit support for events, e.g., Event Calculus [23]. Another technique is event algebras, where an event pattern is defined by an expression built recursively from atomic events and algebra operators. This approach is commonly used in languages for active databases, such as Snoop [13, 12], Ode [19] and SAMOS [17, 18], but also in some general, high-level event notification systems [21].

We identify the following important properties for event pattern specification techniques:

- **Sufficient expressiveness** The technique should be rich enough to express any event pattern that might be of interest in the targeted type of system.

- **Efficient implementation** The detection mechanism should have a low overhead in terms of memory and execution time. This is particularly important in embedded and real-time applications, where it is also vital that safe estimates of worst case memory usage and execution time can be derived statically.

- **Well-defined and intuitive semantics** A formal definition reduces ambiguity and facilitates reasoning about the event detection or a system that utilises it. In particular, formal reasoning requires formal semantics. To be useful in practice, the semantics should be as simple and intuitive as possible.

These properties are relatively straightforward to achieve in isolation. Many existing approaches, in particular those based on temporal logic or similar formalisms, are highly expressive and provide operators with intuitive properties, but this generally means less efficient implementation. Similarly, techniques defined in terms of finite state machines trivially ensures limited resource requirements, but complex procedural pattern definitions are typically difficult to reason about compared to declarative definitions. In particular, composing procedural definitions sometimes give non-intuitive results.

For event algebras, the efficiency issue is often addressed by event contexts. Each operator is given a simple declarative meaning such as *”an A event followed by a B event”* for the sequence operator. In addition to this, a number of contexts are defined, each specifying a certain subset of the simple operator semantics by stating, e.g., whether constituent events may be reused in multiple detections, if detections may overlap, etc. Thus, each operator-context pair can be seen as a separate operator.

The property of intuitive semantics can be supported by supplying algebraic laws that comply with the intuitive interpretation of the algebra operators. Galton and Augusto [16] argues that this is easier to achieve with an algebra semantics based on intervals rather than single time instants. However, their results do not extend trivially to event contexts.

2 Problem Formulation

The main goal of this project is the development of an event algebra that (i) complies with algebraic laws that intuitively ought to hold for the operators, and (ii) permits an efficient implementation with limited resource requirements.

This problem statement is motivated by resource-conscious applications such as real-time and embedded systems. This type of systems require that bounds for memory usage and execution time can be statically determined. Furthermore, they often appear in safety-critical applications for which formal verification is required. Providing laws that the algebra conforms to allows reasoning on a high level of abstraction, and facilitates verification.

We also want to investigate how this algebra can be integrated in existing event-based frameworks. From the area of event-based system design we consider a high-level language for component based development of embedded vehicular systems, and study how this can be extended with the event algebra. We also intend to investigate the impact on implementation level when some activities within a real-time system is triggered by event patterns defined by the algebra, focusin mainly on real-time specific aspects such as scheduling.

3 Contributions

The main contributions of the thesis can be summarised as follows.

- A novel event algebra with well-defined algebraic properties that intuitively ought to hold for the algebra operators. These properties facilitate formal as well as informal reasoning about the algebra and the behaviour of a system that uses it.
- A detection algorithm for the algebra that correctly detects any expression with bounded memory. The algorithm is formally verified with respect to correctness and complexity.
- Schedulability and scheduling theory for systems where some parts are triggered by complex event patterns, and a method to transform such a mixed tasksets into a corresponding set of event-triggered tasks.
- An extention of a software component framework in which components can be triggered by complex event patterns. Moving the responsibility of event pattern detection from components to the framwork permits more general components and thus improves component reusability.

4 Thesis Outline

The thesis will be organised as a monograph titled *”A Constant-memory Event Algebra with Intuitive Algebraic Properties”*, with the following contents:

1. Introduction

   This section states the key objectives, describes the approach by which the objectives have been investigated, and summarises the main contributions of the work.

2. Background and related work

   The section addresses the relevance of the work, and surveys related work in event detection.

3. The event algebra

   This section defines the algebra semantics and presents a number of algebraic properties.

4. Detection algorithm

   An algorithm that implements the event algebra is presented together with proofs of correctness and complexity results.

5. Scheduling theory

This section describes how scheduling properties such as minimum inter-arrival time can be derived for expressions. It also discusses how the scheduling of expression-triggered activites can be combined with existing scheduling methods.

6. Expression-triggered software components

An extension of the component model SaveCCM is presented that allows components to be triggered by complex event patterns.

7. Expression-triggered tasks

This section presents a prototype tool that illustrates the theoretical concepts in earlier sections. The tool performs schedulability analysis on systems where some tasks are expression-triggere, and transforms them into ordinary event-triggered tasksets to suit an existing real-time operating system.

8. Conclusions

The main contributions are summarised and the possible directions of future research are outlined.

5 Publications Related to the Thesis

An Interval-based Algebra for Restricted Event Detection
Jan Carlson, Björn Lisper
In Proceedings of the 1st International Workshop on Formal Modeling and Analysis of Timed Systems (FORMATS'03), Marseille, France, September, 2003

The paper presents an early version of the algebra. The temporal restriction construct is not present, and two different restriction policies are used (one for sequences and one for the remaining operators). No general resource bounds are presented, and the algebraic properties are weak compared to later versions.

An Improved Algebra for Restricted Event Detection
Jan Carlson, Björn Lisper
MRTC report, Mälardalen University, February, 2004

This paper introduces temporal restriction, but for sequences only. For expressions where every sequence has a finite temporal restriction, limited memory requirement is ensured.

An Event Detection Algebra for Reactive Systems
Jan Carlson, Björn Lisper
MRTC report, Mälardalen University, March, 2004

This paper defines the current algebra and a detection algorithm with bounded memory for many, but not all, expressions.

An Intuitive and Resource-Efficient Event Detection Algebra
Jan Carlson
Licentiate Thesis, Mälardalen University, June, 2004

The licentiate thesis presents the algebra, a detection algorithm, and an optimisation algorithm that allows a large class of expressions to be detected with limited memory. It also describes a prototype implementation in Java.

An Event Detection Algebra for Reactive Systems
Jan Carlson, Björn Lisper
In Proceedings of the 4th ACM International Conference on Embedded Software (EMSOFT'04), Pisa, Italy, September, 2004

A short version of the MRTC report where formal proofs are ommitted.

An Event Algebra Extension of the Triggering Mechanism in a Component Model for Embedded Systems
Jan Carlson, Mikael Åkerholm
In Proceedings of the 2nd International Workshop on Formal Foundations of Embedded Software and Component-Based Software Architectures (FESCA'05), Edinburgh, April, 2005

This paper describes how SaveCCM, a component model intended for embedded vehicular systems, can be extended by the event algebra. The extension allows components to be triggered by complex event patterns, in addition to clock signals or single external events.

6 Additional Publications

Value Based Overload Handling of Aperiodic Tasks in Distributed Offline Scheduled Real-Time Systems
Tomas Lennvall, Jan Carlson, Gerhard Fohler
MRTC report, Mälardalen University, May, 2001

Value Based Overload Handling of Aperiodic Tasks in Offline Scheduled Real-Time Systems
Jan Carlson, Tomas Lennvall, Gerhard Fohler
In Proceedings of the Work-in-progress Session, 13th Euromicro Conference on Real-Time Systems, Delft, The Netherlands, June, 2001

Simulation Results and Algorithm Details for Value Based Overload Handling
Jan Carlson, Tomas Lennvall, Gerhard Fohler
MRTC report, Mälardalen University, May, 2002

Languages and Methods for Specifying Real-Time Systems
Jan Carlson
MRTC report, Mälardalen University, August, 2002

Enhancing Time Triggered Scheduling with Value Based Overload Handling and Task Migration
Jan Carlson, Tomas Lennvall, Gerhard Fohler
In Proceedings of the 6th IEEE International Symposium on Object-oriented Real-time Distributed Computing (ISORC'03), Hakodate, Japan, May, 2003

SaveCCM: An Analysable Component Model for Real-Time Systems
Jan Carlson, John Håkansson, Paul Pettersson
In Proceedings of the 2nd International Workshop on Formal Aspects of Component Software (FACS'05), Macao, October, 2005

7 Graduate Courses

The PhD degree requirement of passed courses of at least 50 credits is fulfilled.

Course	**Credits**	**Status**
Computation II	2.5	Completed
Logic II	2.5	Completed
Advanced Functional Languages	5.0	Completed
Advanced Real-time Systems	5.0	Completed
Software Engineering	10.0	Completed
Advanced Type Systems	5.0	Completed
Formal Specification of Real Time Systems	1.0	Completed
Discrete Structures II	2.5	Completed
Summer School on Advanced Functional Programming	2.0	Completed
Concurrency Theory and Time	3.0	Completed
Formalisms, Algorithms and Tools in Formal Methods for Real-Time	3.0	Completed
Real-Time and Embedded Systems	1.0	Completed
Program Analysis	5.0	Completed
Semantics Specification of Types and Programming Languages	4.0	Near completion
Total credits	**51.5**	

8 Remaining Work and Timeplan

The event algebra and the detection algorithm are finished, but some more work is needed for the formal proofs of correctness and complexity results. The remaining theoretical part is to formulate the scheduling theory, including the derivation of scheduling properties and how to combine it with existing sheduling methods. Additionally, a prototype tool should be implemented where systems with expression-triggered tasks can be specified, analysed, and transformed into systems of ordinary event-triggered tasks.

8.1 Planned Publications

- A tecnical report defining the new bounded detection algorithm, together with and proofs of correctness and complexity.
- A journal article summarising the theoretical work on the algebra and the detection algorithm.
- A conference or workshop paper presenting the scheduling theory and the concept of expression-triggered tasks.

8.2 Timeplan

Planned research activity during the remaining time:

Period	**Activity**
2006 Jan–Jun	75%
2006 Jul–Dec	25%
2007 Jan–Feb	80%

Milestones:

Deadline	**Milestone**
2005 Dec	Thesis proposal presented Correctness and complexity proofs finalised
2006 Jan	Technical report published
2006 Feb	Scheduling theory finished
2006 Apr	Journal paper submitted
2006 Jun	Prototype tool design and implementation done
2006 Dec	Thesis draft ready for review
2007 Feb	Thesis finished and defended

References

[1] Jan Carlson. Languages and methods for specifying real-time systems. Technical report, Mälardalen Real-Time Research Centre, Mälardalen University, August 2002.

[2] Jan Carlson. An intuitive and resource-efficient event detection algebra. Licentiate thesis No. 29, June 2004. Mälardalen University, Sweden.

[3] Jan Carlson and Mikael Åkerholm. An event algebra extension of the triggering mechanism in a component model for embedded systems. In *Formal Foundations of Embedded Software and Component-Based Software Architectures*. ENTCS, 9 April 2005.

[4] Jan Carlson, John Håkansson, and Paul Pettersson. SaveCCM: An analysable component model for real-time systems. In *Proceedings of the 2nd Workshop on Formal Aspects of Components Software (FACS 2005)*, Electronic Notes in Theoretical Computer Science. Elsevier, 2005.

[5] Jan Carlson, Tomas Lennvall, and Gerhard Fohler. Value based overload handling of aperiodic tasks in offline scheduled real-time systems. In *Work-in-progress Session, 13th Euromicro Conference on Real-Time Systems*, Delft, The Netherlands, June 2001.

[6] Jan Carlson, Tomas Lennvall, and Gerhard Fohler. Simulation results and algorithm details for value based overload handling. Technical report, Mälardalen Real-Time Research Centre, Mälardalen University, May 2002.

[7] Jan Carlson, Tomas Lennvall, and Gerhard Fohler. Enhancing time triggered scheduling with value based overload handling and task migration. In *6th IEEE International Symposium on Object-oriented Real-time distributed Computing*, Hakodate, Japan, May 2003.

[8] Jan Carlson and Björn Lisper. An interval-based algebra for restricted event detection. In Kim G. Larsen and Peter Niebert, editors, *First International Workshop on Formal Modeling and Analysis of Timed Systems (FORMATS 2003)*, volume 2791 of *Lecture Notes in Computer Science*, pages 121 – 133. Springer-Verlag, 6–7 September 2003.

[9] Jan Carlson and Björn Lisper. An event detection algebra for reactive systems. Technical Report MDH-MRTC-117/2004-1-SE, Dep. of Computer Science and Engineering, Mälardalen University, Sweden, April 2004. Available from: http://www.mrtc.mdh.se.

[10] Jan Carlson and Björn Lisper. An event detection algebra for reactive systems. In *Proceedings of the 4th ACM International Conference on Embedded Software (EMSOFT)*. ACM, New York, 27–29 September 2004.

[11] Jan Carlson and Björn Lisper. An improved algebra for restricted event detection. Technical Report MDH-MRTC-159/2004-1-SE, Dep. of Computer Science and Engineering, Mälardalen University, Sweden, February 2004. Available from: http://www.mrtc.mdh.se.

[12] 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.

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

[14] Christophe Dousson. Alarm driven supervision for telecommunication networks: II- On-line chronicle recognition. *Annals of Telecommunications*, pages 501–508, October 1996. CNET, France Telecom.

[15] Patrick Th. Eugster, Pascal A. Felber, Rachid Guerraoui, and Anne-Marie Kermarrec. The many faces of publish/subscribe. *ACM Comput. Surv.*, 35(2):114–131, 2003.

[16] A. Galton and J. C. Augusto. Two approaches to event definition. In *Proc. of Database and Expert Systems Applications 13th Int. Conference (DEXA'02)*, volume 2453 of *Lecture Notes in Computer Science*. Springer-Verlag, September 2002.

[17] 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.

[18] 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.

[19] 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.

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

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

[22] Hermann Kopetz. Event-triggered versus time-triggered real-time systems. Research Report 8/1991, Technische Universität Wien, Institut für Technische Informatik, Treitlstr. 1-3/182-1, 1040 Vienna, Austria, 1991.

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

[24] Sacha Krakowiak. What is middleware, 2003. `http://middleware.objectweb.org`.

[25] Tomas Lennvall, Jan Carlson, and Gerhard Fohler. Value based overload handling of aperiodic tasks in distributed offline scheduled real-time systems. Technical report, Mälardalen Real-Time Research Centre, Mälardalen University, May 2001.

[26] C. Liebig, B. Boesling, and A. Buchmann. A notification service for next-generation it systems in air traffic control. In *GI-Workshop: Multicast-Protokolle und Anwendungen*, Braunschweig, Germany, May 1999.

[27] 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.

[28] Peter R. Pietzuch. *Hermes: A Scalable Event-Based Middleware*. PhD thesis, University of Cambridge, February 2004.