Operator Patterns for Analysis of Composite Events in Timed Automata

AnnMarie Ericsson, Robert Nilsson, Sten F. Andler

Department of Computer Science
University of Skövde
Box 408, 541 28 Skövde, Sweden
e-mail: {ammi.ericsson, robert.nilsson, sten.f.andler}@ida.his.se

***Abstract*— Event-triggered real-time systems interact with the environment by executing actions in response to monitored events. Such systems may be implemented using event condition action (ECA) rules, which execute an action if the associated event occurs and a specified condition is true. However, the ECA rule paradigm is known to be hard to analyze with respect to correctness and timeliness, which is not conducive to the high predictability requirements typically associated with real-time systems. To still take advantage of the ECA rule paradigm when event-triggered real-time systems are developed, we propose an approach where systems are specified and analyzed in a high-level formal language (timed automata) and later transformed into the ECA rule paradigm. We especially focus on a high-level approach for specifying and analyzing composite event occurrences in timed automata.**

I. Introduction

Event-triggered real-time systems respond to occurrences of events by executing actions associated with the event occurrence. This behaviour is convenient to implement using event condition action (ECA) rules [1]. In the ECA rule paradigm, actions are executed as a response to an event occurrence if a specified condition is true. ECA rules are used to specify the behaviour of active databases, where the reactive capability is managed inside the active database management system instead of being spread among several applications [2]. The event part of ECA rules may be primitive or composite. A primitive event is an atomic occurrence while a composite event is a combination of primitive or composite events.

The use of an active real-time database system (ARTDBS) such as DeeDS [3], makes it possible to implement event-triggered control applications that respond to sporadic event occurrences in a timely manner, as a set of ECA rules. Besides the benefit that applications can react to external events in a timely manner, the use of an ARTDBS adds facilities such as backward recovery, concurrency control and persistent storage of data about the controlled environment.

One of the main disadvantages of ECA rules is the poor CASE-tool support for development activities specialized on ECA rule applications [2]. ECA rules is a low-level specification language and rules may depend on each other in many intricate ways, which makes it hard to predict what impact a change of a single rule has on the entire rule set.

To facilitate the development and maintenance of active real-time applications, we propose a method which simplifies the analysis of ECA rule sets and allows us to verify their temporal correctness. The approach is to raise the abstraction level by specifying and analyzing systems in a formal specification which is later transformed into ECA rules. Our approach is similar to a compiler approach [4], however, starting from a formally verified specification including time constraints. The formal specification of our choice is timed automata, which are finite automata extended with a set of clocks [5].

Timed automata are designed to be a specification language for real-time systems and we propose a new method for how to specify and transform timed automata models to ECA rules. The timed automata specifications can be modelled and verified in the CASE tool UPPAAL [6]. The use of UPPAAL makes it possible to graphically simulate the behaviour of the model as well as automatically perform model checking and reachability analysis.

One of the strengths of ECA rules is their ability to respond to sporadic event occurrences arriving in arbitrary order. This behaviour is not as straightforward to specify in timed automata as in ECA rules. We therefore propose timed automata *operator patterns* as a general solution for specifying and analyzing composite events that can capture sporadic event occurrences in timed automata. We believe that the use of operator patterns will facilitate development of timely active applications using an expressive rule language, especially if the patterns are integrated with some of the CASE tools available for timed automata.

II. Generate ECA rules

As previously mentioned, there are several benefits of raising the abstraction level and specifying reactive applications as timed automata instead of directly in ECA rules. Being able to rely on the correctness of transformations from the timed automata to corresponding ECA rules is crucial for the usefulness of this approach. Equally important is the ability to conveniently express reactive behaviour in the timed automata model. For reactive behaviours involving the monitoring of primitive event occurrences in deterministic environments, we have proposed [7] a method for transforming timed automata to rules: Entering an external state in the timed automaton is transformed to a primitive event occurrence in the rule base;

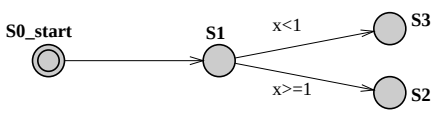

Fig. 1. Conjunction in recent context

Guards are transformed to conditions; Transition assignments are transformed to procedures which are executed in the action part in the resulting rule base.

The method works properly if there is always a deterministic choice of next transition to take from a state in the automaton. As an example the timed automaton in Figure **??** will be transformed to the following pair of ECA rules:

On $E_{arrivetoS1}$
If $x >= 1$
Do actionS1toS2()

On $E_{arrivetoS1}$
If $x < 1$
Do actionS1toS3()

When specifying reactive behaviours involving the monitoring of combinations of multiple events, however, the specification of the automaton as well as the translation process to ECA rules becomes much more complex. For this purpose, the following sections introduce the concept of timed automata operator patterns where occurrences of events, which are easily transformed to composite events in the rule set, are specified in timed automata.

III. Timed automata operator patterns

A set of event occurrences may be combined to a composite event using operators like for example conjunction, disjunction, sequence and negation. A timed automaton operator pattern is a small timed automata specification representing the behaviour of a composite event combined by a certain operator in a specific context [8]. The exact behaviour of operator patterns depend on the execution model of the target ARTDBS and in particular on the current context [8]. The context decides which instances of event occurrences the composite event should be combined of. If recent context is used, the most recent occurrences of each constituent event are used to form composite events. If chronicle context is used, event occurrences are consumed in chronicle order, and the oldest unused occurrences of each constituent event are used to form a composite event occurrence.

The use of timed automata operator patterns makes the divergence in complexity and behaviour of operators in different contexts obvious. As an example, we discuss the operator pattern for conjunction in both recent and chronicle context. Additional operators are discussed in previous work [7] together with initial proofs for equivalence of semantics between operator patterns and ECA rules.

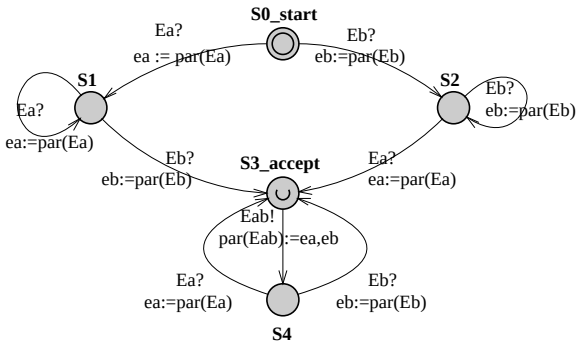

Fig. 2. Conjunction in recent context

A. Assumptions

The environment interacting with the modelled system is assumed to be modelled in a separate automaton which is synchronizing with the composite event pattern as an event occurs. If for example the occurrence of event type E_a represents that a sensor is active, the occurrence raising the event that the sensor becomes active must synchronize with the composite event pattern on channel Ea in the operator model.

In the following timed automata examples, exclamation mark denotes send on a channel and question mark denotes receive. Automata synchronizing on send/receive channel must take their synchronized transitions simultaneously.

B. Conjunction in recent context

A composite event type may be combined by different operators, like for example conjunction, disjunction or sequence. Figure 2 shows a timed automata conjunction pattern for recent context, which is an automaton that signals an event occurrence of type E_{ab} as an event of type E_a *and* E_b have occurred in recent context. When the operator pattern is in its starting state S0, there are no unused occurrences of type E_a or E_b in the event history. When there is an occurrence of both type E_a and E_b in the event history, a composite event of type E_{ab} is raised. In the operator pattern, the occurrence of event type E_{ab} causes that the acceptance state S3 is reached and the model is ready to synchronize with some other automaton waiting for the operator pattern to synchronize on channel Eab!.

The variables *ea* and *eb* represents event parameters associated with event type E_a and E_b. The paramters must be saved since they will be associated with the composite event occurrence. In recent context, only the most recent event parameters are interesting, since they belong to the most recent event occurrences. This means that a single buffer is sufficient to save the parameters. When a new event occurs, the old event parameters are simply overwritten and as the composite event

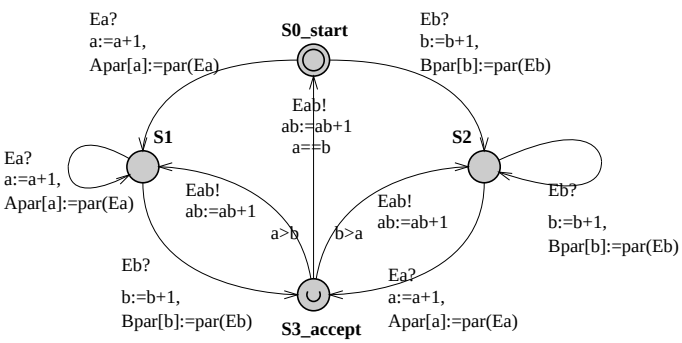

Fig. 3. Conjunction in chronicle context

E_{ab} is raised, the parameters associated with the most recent event occurrences are *ea* and *eb*.

C. Conjunction in chronicle context

In Figure 3 the counters *a*, *b* and *ab* are used to count the number of event occurrences of type E_a, E_b and E_{ab}. Since this is a model where no event occurrences are invalidated, the number of used event occurrences of each type are equal to the number of composite event occurrences (ab) as the system is in state S3.

In chronicle context, the oldest unused event occurrence is used to form composite events, which means that all unused event parameters must be stored. To model this, the operator pattern for conjunction uses the arrays *Apar* and *Bpar* to store parameters associated with events. As a composite event of type E_{ab} occurs, it will have access to its constituent events parameters which will be stored in Apar[ab] and Bpar[ab].

IV. Expiration times

Since the patterns are to be used in real-time systems, it is of high interest to add expiration times to the patterns. This is because a composite event may be useless after a certain amount of time, since for example the deadline for its associated action part may already have expired. The operator patterns can be extended to take expiration times into account by saving timestamps for each event occurrence and add transitions which invalidate event occurrences as their valid time has expired.

The operator pattern in Figure 4 is simular to the operator pattern in Figure 3, but it also saves time stamps of events as parameters, to model expiration times. In Figure 4 the expiration time for composite events is set to 10, the counters *ca* and *cb* are used to count the number of occurred events of type E_a and E_b and the counters *a* and *b* counts the number of *used* event occurrences of type E_a and E_b. In a scenario where an event of type E_a occurs as the pattern is in state S0, the time of the event occurrence is saved in *Apar[ca]*. If no event of type E_b occurs which can be matched with the occurrence of type E_a within 10 time units, the automaton takes the transition back to S0 if there are no other unused

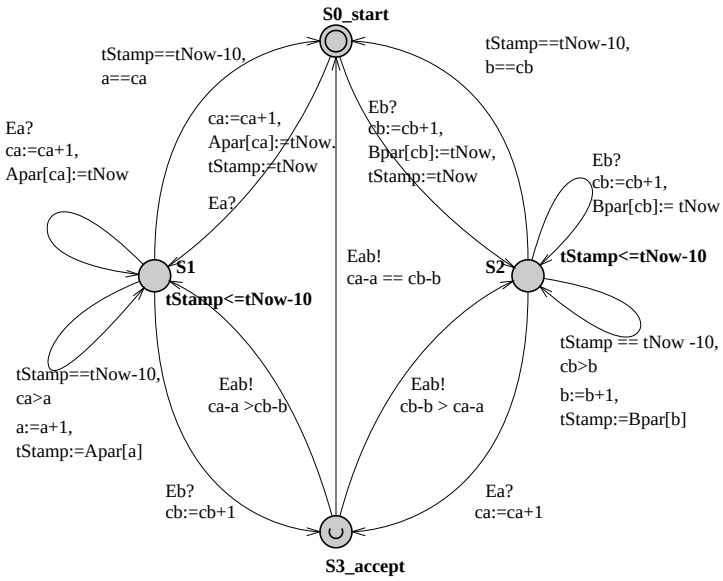

Fig. 4. Conjunction in chronicle context with expiration times

occurrences of type E_a in the event history. If there are unused occurrences of type E_a in the event history, a new expiration time is set. Additional work on expiration times in operator patterns are presented in [7].

V. Combination of patterns

Timed automata operator patterns may be combined to analyze and verify larger and more complex combinations of event occurrences. The Ea and Eb channels may synchronize with the acceptance channel (Eab in the example) of another operator pattern modelling e.g. conjunction, disjunction or sequence of other event occurrences. This means that the behaviour of complex combinations of composite event occurrences in different contexts can be automatically analyzed in a CASE-tool before they are implemented in an ARTDBS.

VI. Extended timed automata

To facilitate the work of specifying and analyzing rule based systems, we suggest an integration of operator patterns in existing timed automata CASE-tools.

Instead of specifying the entire operator pattern, a single transition with for example the label $E_a \triangle E_b\langle recent\rangle$ could represent the acceptance of E_a and E_b in recent context (notation previously suggested by [9]). In this way, the operator pattern can be transparent to the specifier, only used in the verification process of the specified system.

VII. Related work

The problem of analyzing a set of ECA rules is well known and has been the subject of a lot of previous research. However, a complete prediction of the interactions in a set of ECA rules is identified by both [10], and [4] to be an undecidable problem. It is impossible to assert with certainty whether an

harmful interaction between rules will take place considering all database states and all possible action statements. This is why the approach to solving this problem has to contain some restriction, or simplification of the problem, for example to exclude the state of the database in the analysis, or to raise the abstraction of the analysis, as performed in [4],[10],[11],[12].

In our approach we raise the abstraction level since the system is first specified in a higher level specification language. This approach is previously taken by for example [12] who has created a compiler for this purpose. The novelty of our approach compared to [12] is to include time constraints and to formally verify the high-level model before its behaviour is transformed into rules. The use of a formal approach, assuming that the transformation to rules maintains the verified behaviour, makes it possible to analyze the set of rules on a formal specification level, using for example model checking techniques.

VIII. Conclusion

The effort spent on developing, maintaining and extending active applications will be reduced by our approach. The correctness of applications is analyzed before rules are created, and the effect of a change in a rule can be analyzed in the formal specification before changes are integrated in the system. This will simplify the task of identifying worst-case scenarios with respect to rule triggering, which is essential in attaining predictable systems. However, even if rule triggering can be analyzed formally, some sources of non-determinism cannot be avoided during the analysis phase. For example differences in execution times of actions, or race-conditions. Hence, the analysis of the rule sets must be combined with timeliness testing to gain confidence in the overall correctness of an event-triggered system. To support this, the approach of automatically generating active applications from formal specifications allows for design for testability to be integrated transparently into the system.

The presented operator patterns are a step towards an automatic method for transforming timed automata specifications to ECA rules. Our intention is to continue this work of refining the operator patterns and the method briefly outlined in this paper. The use of operator patterns makes it possible to further enhance the abstraction level as active applications are developed. The main aim of our work is to realize a CASE-tool solution where development of rule based applications are specified in a high-level formal specification before the model is automatically transformed into the ECA rule paradigm with maintained semantic.

References

[1] K. Ramamritham, "Real-time databases," *International Journal of Distributed and Parallel Databases*, vol. 1, no. 2, pp. 199–226, 1993.

[2] N. W. Paton and O. Diaz, "Active database systems," *ACM Computing Surveys*, vol. 31, no. 1, pp. 63–103, 1998.

[3] S. F. Andler, M. Berndtsson, B. Eftring, J. Eriksson, J. Hansson, and J. Mellin, "DeeDS: A distributed active real-time database system," University of Skövde, Sweden, Tech. Rep. HS-IDA-TR-95-008, 1995.

[4] E. Falkenroth, "Database technology for control and simulation," Ph.D. dissertation, Linköping University, Sweden, 2000.

[5] R. Alur and D. L. Dill, "A theory of timed automata," *Theoretical Computer Science*, vol. 26, pp. 183–235, 1994.

[6] K. G. Larsen, P. Pettersson, and W. Yi, "UPPAAL in a nutshell," *International Journal on Software Tools for Technology Transfer*, vol. 1, no. 1-2, pp. 134–152, 1997.

[7] A. Ericsson, "Verifying transformations between timed automata specifications and ECA rules," Master's thesis, University of Skövde, Sweden, 2003.

[8] S. Chakravarthy and D. Mishra, "Snoop: An expressive event specification language for active databases," *Data Knowledge Engineering*, vol. 14, no. 1, pp. 1–26, 1994.

[9] J. Mellin, "Predictable event monitoring," Licentiate thesis, Linköping University, Sweden, Tech. Rep. 737, 1998.

[10] A. Vaduva, "Rule development for active database systems," Ph.D. dissertation, University of Zürich, 1999.

[11] M. Berndtsson, S. Chakravarthy, and B. Lings, "Extending database support for coordination among agents," *International Journal on Cooperative Information Systems*, vol. 6, no. 3–4, pp. 315–339, 1997.

[12] E. Falkenroth and A. Törne, "How to construct predictable rule sets," in *Advance proceedings of the joint 24th IFAC/IFIP workshop on real time programming and 3rd international workshop on active and real-time database system*, 1999, pp. 33–40.