Enabling Tool Support for Formal Analysis of Predictable sets of ECA Rules

Thesis proposal

AnnMarie Ericsson
annmarie.ericsson@his.se

University of Skövde
October 2006

Abstract

This thesis proposal addresses support for utilizing an existing formal analysis tool when predictable rule-based systems are developed. One of the main problems of the rule-based paradigm is that it is hard to analyze the behavior of rule sets, which is conflicting with the high predictability requirements typically associated with real-time systems. Further, analysis tools developed for rule-based systems typically address a specific platform or a specific part of the development chain.

In our approach, rules are initially specified in a high-level language. We enable a powerful analysis tool not designed for rule based development, to be utilized for analyzing the rule base. This is done by transforming the set of rules, with maintained semantics, to a representation suitable for the target analysis tool. Our approach provides non-experts in formal methods with the ability to formally analyze a set of rules.

Contents

1 Introduction

1.1 The problem

Event-triggered real-time systems respond to occurrences of events within a predictable period of time. Such systems are typically idle waiting for an event, for example caused by an object passing a sensor, to occur. When an event occurs, a message is sent to the interior of the system where the message is processed and an output produced [25].

The behavior of an event triggered system can be implemented as event condition action (ECA) rules [31]. In the ECA rule paradigm, actions are executed as response to event occurrences if their associated conditions are true. ECA rules may, for example, be used for describing the behavior of logic control applications [3], or for specifying the behavior of a system using an active database, where the reactive capability is managed inside the active database management system instead of being spread among several applications [29].

Developing applications specified as ECA rules is complicated. This mainly depends on the rules ability to interact with each other. In rule based systems, execution of one rule can, for example, trigger another rule, starting a chain of rule triggerings, or changing the outcome of the condition evaluation of a third rule. Interactions between rules makes a rule set hard to analyze and maintain since changing, adding or removing one rule may have consequences that are hard to predict.

Additionally, a set of ECA rules may behave differently on different platforms depending on the execution model (the way rules are processed during execution) of the platforms used. If, for example, different priorities are specified for the rules in rule set R_{set}, executing R_{set} on a target platform *with* support for priorities may result in a different behavior compared to executing R_{set} on a target platform *without* support for priorities. This implies that knowledge about the target platform must be included in, for example, rule analysis [18].

Real-time systems have requirements on timeliness, meaning that the system must be sufficiently efficient and have predictable resource requirements. A timely system meet its deadlines, implying that applications must be thoroughly analyzed and that time in the form of, for example, execution times and deadlines, must be included in behavior analysis.

To enhance the level of analysis, formal methods can be used for analyzing critical parts of a system [12]. However, despite previous research in formalizing different aspects of active databases [28], formal techniques are still rarely exploited during implementation of rule-based systems.

1.2 Approach

Our approach for development of predictable rule-based systems is to utilize the strength of the existing analysis tool Uppaal which is designed for specifying and verifying timed-automaton models. Rules will be specified in the novel tool prototype REX with support for designing rules, transforming infor-

mation about rules to UPPAAL and generate executable rules to different target platforms.

1.3 Limitations of other approaches

The CASE tool support for analysis of rule based systems is currently immature. Several tools exists, for example, Peard[23], TriGS[24] and the tools within the IDEA project[16], each of them solving a sub part of the development chain [23], tailored for a specific target platform [24], or specific execution model, with limited or no ability to communicate with other tools.

Additionally, as far as we know, no previous CASE tool for rule based systems supports formal analysis of a set of rules with composite events and time constraints.

1.4 Contributions and limitations

The expected main contribution of this project is an approach for formally analyzing a set of rules with real-time constraints utilizing an existing analysis tool. The approach will be demonstrated in a development tool prototype, REX, which will be developed within this project.

Additionally, the project is expected to contribute with a specification of what information is needed when sets of real-time rules and information about their processing during runtime is exchanged between tools and how this information can be represented.

Further, our work will enhance the support for formal analysis of rule based systems, particularly rule-based systems with time constraints, by providing support for automatically analyzing rules utilizing the model checker in UPPAAL[10].

In this work, the expressiveness of the rules condition part that is possible to *analyze* will be limited to Boolean values. The UPPAAL analysis will not support, for example, SQL expressions in conditions. On the other hand, analysis of composite events will be supported, increasing the expressive power of rules. Moreover, by focusing on composite events, the contribution of our work is also useful in the area of complex event processing since the correctness of composing events are interesting even if the events are not attached to a rule [21].

This thesis proposal is organized as follows: Section 2, 3 and 4 provide background information about event composition and UPPAAL. In Section 5, a motivation for the thesis is given while Section 6 contains Thesis statement in form of aim and objectives. Section 7 present an overview of the current status of the project while in depth results are presented in Section 8,10 and Appendix. The remaining section contain discussion and conclusion.

2 ECA rules

Reactive mechanisms were introduced in the late '80s in the form of Event-Condition-Action rules (ECA-rules) in active databases.

During the '90s concrete systems for combining active database capability with real-time constraints emerged and active real-time databases such as Polyhedra[30], DeeDS[6] and REACH[14] evolved. Today, several commercial systems and standards, for example, Oracle, Polyhedra[30] and SQL3, have support for ECA rules or a more simple form of rules called triggers.

Lately, research on active rules has taken a broader scope than active database systems. Rules are decoupled from the large monolithic database platform, making it possible to use rule-based applications for a larger set of applications that are not dependent on database facilities. ECA rules are, for example, used for specifying control applications[3] and for specifying the behavior on local nodes for the semantic web[2]. For nodes to be able to communicate with each other, RuleML[2], an XML based markup language for rules are under development.

2.1 Knowledge and execution model

A set of ECA rules can be described by its knowledge and execution models[29]. The knowledge model describes what can be said about a set of ECA rules and the execution model describes how a set of rules is processed during execution. The knowledge and execution models are thoroughly described in Paton and Diaz survey [29]. In the following subsections, concepts needed as background knowledge in this proposal is given.

The relative time of triggering, condition evaluation and action execution depend on the Event-Condition (and Condition-Action) *coupling mode* of the executing platform. In immediate coupling mode, the condition is evaluated (action executed) immediately after the event occurrence (condition evaluation). In deferred coupling mode the condition evaluation (action execution) is performed in the same transaction but not necessarily immediately. In detached coupling mode, the condition evaluation (action execution) is performed in another transaction[29].

The *transition granularity* is the relation between occurred events and triggered rules. The transition granularity is named *tuple* if a single event occurrence triggers a single rule and *set* if several event occurrences of the same type in the same transaction together triggers a single rule[29].

2.2 Confluence and termination

Two important characteristics of a rule based system is termination and confluence. Confluence concerns whether the result of executing a set of simultaneously triggered rules are dependent on the execution order or not. The scheduling phase of rule execution determine what happens if several rules are triggered at the same time. The order in which simultaneously triggered rules are executed may affect the resulting outcome of rule execution. A possible solution for solving this problem is to provide support for prioritizing the rules.

A set of rules may have a non-terminating behavior if rules are triggering each other in a cyclic order, for example, if the execution of rule R1 triggers rule R2 and the execution of rule R2 triggers rule R1.

Previous research has been performed on analyzing termination and confluence on rule based systems (e.g. [1, 7, 8]), however, analysis tool support for automatically verifying non-termination and confluence is still weak.

3 Composite events

A primitive event occurrence happens at a point in time and is raised by a single occurrence, for example, an update of a database or the triggering of a sensor. A composite event is raised by a combination of primitive or composite events. The *initiator* of a composite event is the event initiating the composite event occurrence and the *terminator* is the event terminating the composite event occurrence.

3.1 Consumption modes

Composite events are combined by operators such as disjunction ($\bigtriangledown$), conjunction ($\bigtriangleup$) and sequence ($;$). Events contributing to a composite event occurrence may carry parameters, for example, from the activity causing the event. Since parameters carried by event occurrences of the same type may be different, it is important to consume events of the same type in a predefined consumption policy[17].

Some frequently used consumption policies are recent and chronicle. In the recent consumption policy, the most recent event occurrences of contributing event types are used to form the composite event occurrence and in the chronicle consumption policy, the earliest unused event occurrences are used. The recent consumption policy are, for example, useful if calculations must be performed on combinations of the last measured values of temperature and pressure in a tank. The chronicle consumption policy is, for example, useful if sensors are placed along a conveyor-belt monitoring objects travelling along the belt and combinations of sensor events triggered by the same object is needed. In that case events must be combined in occurrence order since the first event from the first sensor and the first event from the second sensor are likely triggered by the same object.

3.2 Expiration times

For event composition in real-time systems, we need to extend composite events with expiration times. This is because composite events may become useless after a certain amount of time as, for example, the deadline for its associated action part already have expired. It is also beneficial to define life span of events in order to clean the system from semi-composed events in an efficient manner [13], memory resources may be limited and should not be wasted on useless information about invalidated event occurrences.

The semantics of expiration times are that the composition of an event is interrupted if the terminator does not occur within a defined period relative

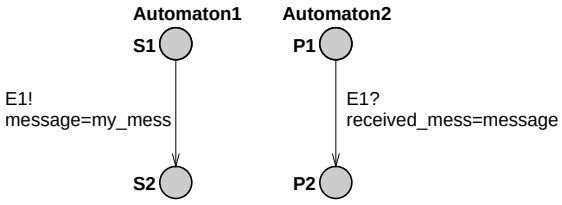

Figure 1: Example of two synchronizing timed-automata.

to the initiator occurrence. Consider, for example, the composite event $E_3 = E_1; E_2$, (an event of type E_3 is generated if an event of type E_1 followed by E_2 is detected). If the event of type E_2 must occur within 10 time units after the event of type E_1 for the composite event E_3 to be interesting for the system, the expiration time for that composite event is 10. In this paper, the syntax used for expressing expiration times are $E_1\langle exp(expirationtime)\rangle; E_2$ [27].

4 UPPAAL

Timed automata are finite automata extended with a set of clocks[5]. UPPAAL is a toolbox for modelling and analyzing specifications built on the theory of timed-automata with additional features. The tool is developed jointly by Uppsala University and Aalborg University[10]. A model in UPPAAL is built by one or more synchronizing timed automata. Each timed automaton simulates a process which is able to synchronize with other automata.

4.1 Specifying models in UPPAAL

Each automaton in UPPAAL contain a set of states S with an initial state $s_0 \in S$. Each state can have constraints on clock values forcing a transition to be taken within a certain time limit. If, for example, c_1 is a clock variable, and the invariant $c_1 <= 4$ is defined in state s_0, the automaton is not allowed to be in state s_0 if $c_1 > 4$.

A transition can be associated with three parts; (i) constraints on clocks and variables specified by a guard g,(ii) reset clock values and change variable values with action a, and (iii) communicate with other automata by synchronizing on global channels.

To send data on channel x (! is the notation for send), another automaton must simultaneously receive the message on the same channel (? is the notation for receive). Synchronizing transitions imply parallel composition of two automata.

In Figure 1, Automaton1 and Automaton2 synchronizes on channel E_1 when Automaton1 is in state S_1 simultaneously as Automaton2 is in state $P1$. During the transition, Automaton1 sends the content of the variable *my_mess* to

Automaton2 by assigning *my_mess* to the global variable *message* read by Automaton2.

4.2 Analyzing models in UPPAAL

Given an UPPAAL model of a system, properties of the model can be checked by specifying the properties in a simplified version of CTL (computation tree logic) [4] and ask UPPAAL to check the specified properties.

The result of querying the model is either that the property is satisfied or not satisfied. The property can quantify over specific states or over a trace of states. It is, for example, possible to ask if a specific variable will always have a specific value in a specific state or if a certain state can be reached within a limited time period [9].

The syntax for describing the property of a process in a state is $P.i$ where P is the name of the process and i is the name of the state in that process (additionally, i can be a variable or a clock in that process). Given a state formula, for example, $P.i < 5$, the path formula $\exists <> P.i < 5$ (E<> is the syntax for $\exists\Diamond$ in UPPAAL) is used to check whether there exists a path where variable i is less than 5 in process P. For an in depth description and tutorial on the capacity of model-checking in UPPAAL we refer to [9].

5 Motivation and related work

The behavior of a set of executing ECA rules is hard to predict due to inter-relationships between rules. When real-time requirements are introduced into the ECA rule paradigm the variety of analysis tasks that needs to be performed during development is extended due to the need of analyzing the systems timely behavior.

The predictability requirement of critical parts of a real-time system may require the use of a formal method for verifying behavioral correctness. However, despite previous research in formalizing different aspects of active databases [28], formal techniques are still rarely exploited during implementation of rule based systems.

In Ray and Ray [32] a method is proposed for reasoning about active database applications using a model checker. An example is shown where a small set of rules is transformed to a representation of a finite automaton. In short, each cell in the database is transformed to two variables in the automaton and each rule is transformed to a boolean variable.

The idea of Ray and Ray [32] is similar with the ideas motivating part of this work, namely utilizing an existing model checker for verifying a set of rules. However, Ray and Ray only address a specific execution model with primitive events stemming from updates of cells in an active database. The transformation from rules to the finite automata is made manually and the resulting finite automata representation is very specific for the current set of rules. In this project a more general approach to the formal verification is taken

where different execution models are considered together with composite events and time constraints.

Several researchers have identified that lack of tools supporting rule based development is a major issue that needs to be addressed [19, 34, 24]. Several tools exist, each of them solving a sub part of the development chain [33, 23], tailored for a specific platform[24] or is a rigorous system [16].

Trigs[24] and PEARD [23] support rule debugging. PEARD is a prototype including means for visualizing rule execution, inserting breakpoints and inspect values of variables in breakpoints. The primary objective for PEARD is to experiment with useful debugging features.

TriGS provides a set of tools in order to support the whole development cycle of an active database application [24]. TriGS debugger have support for visualizing trace data for post execution analysis and interactive rule debugging by means of breakpoints, and a replay and simulate mechanism. TriGS debugger is designed specifically for the TriGS platform.

The most rigorous approach for supporting development of rule-based systems were taken in the IDEA project[16] providing a set of supporting tools for development of rules with primitive events.

The part of this thesis proposal that concerns making rules less platform dependent is related to the work of Wolfgang et al. [26] where concepts for a general rule language is defined enabling rules constructed for enabling different execution models to interact.

6 Thesis statement

This thesis proposal address the problem of developing formally verified rules for rule-based systems with real-time constraints.

A major problem with rule-based systems is the rules ability to affect each other during execution. Additionally, the fact that a set of rules can behave differently on different platforms implies that a set of rules that behave correctly on one platform may have a faulty behavior on another platform.

Existing tools for analyzing and developing rule-based systems are mainly focused on specific target languages or specific parts of the development chain, making them hard to adopt in arbitrary contexts [19, 11]. Further, as far as we know, support for formally verifying rules, particulary with composite events and real-time constraints does not exist.

The lack of standards for execution models implies a need for a development tool adaptive to different execution models since the execution model must be considered during rule analysis. However, it is not clear from previous research what information is needed to describe the execution model of a rule-based platform with real-time constraints and how this information should be represented.

6.1 Aim

The aim of this thesis is to provide support for utilizing an existing timed automata model-checker for performing formal analysis of rules. In particular, development of sets of rules with real-time constraints adaptive for different execution models will be addressed.

6.2 Objectives

The aim of this thesis will be reached by fulfilling the following objectives:

1. **Define the rule language of REX and information needed to tailor the language to a specific execution model** The rule language supported by this development approach will be defined. The language will not be tailored to a specific execution model but limited to the constructs that REX can transform to timed automata. SQL-questions in conditions requiring, for example, aggregations on values in the database, are an example of a type of construct that will not be transformed to timed-automata.

   For developing rules to a specific platform, the expressive power of the rule language must be limited by the execution and knowledge model of the target platform. The execution model affects the behavior of a set of rules, implying that the execution model must be considered when a set of rules is transformed to a different representation for behavioral analysis.

2. **Develop XML-schema and algorithms for transformation**
   An XML schema will be developed and used for representing the set of

rules and as a source language when rules are transformed to tools and platforms.

All constructs in the rule language, limited by the current target platform and analysis tools to be used, must be mapped and transformed from the XML notation to i) constructs in the analysis tools to be utilized, ii) constructs in the executable representation of the target platform.

Algorithms for transforming rules from the novel XML notation to different representations will be developed for the target platform Polyhedra and the analysis tool UPPAAL. The task of developing algorithms for transforming rules from the XML-notation to additional platforms, such as ruleCore[33] and DeeDS[6] is optional.

3. **Investigate how to perform model checking on the rule set**
The set of rules will be transformed to the timed automata analysis tool UPPAAL[10]. However, to be able to utilize the model-checker in UPPAAL for verifying properties of the rule set, for example, to specify the property *"an event of type E_1 is always generated before an event of type E_2"* requires knowledge about both the CTL language used for property specification in UPPAAL as well as in depth knowledge about the timed automaton representation of the set of rules.

   According to Dwyer et al.[20], most of the verification properties specified for model-checking finite automata specifications are variations of a small set of property specifications. Assuming that this is also true for rule based systems, a set of high-level property patterns tailored for rule based systems will be defined in a language easy to use by non-experts in formal methods. Algorithms will be defined for transforming the instantiated property patterns to a correct property specification in CTL.

4. **Tool support for transformation process**
Transforming information about rules and rule processing to timed automata is an error-prone and time consuming task. To prevent transformation errors from corrupting the rule base, we need a new rule development tool for automating the process of transforming sets of rules to tools, such as UPPAAL, and platforms as well as for obtaining semantic information from tools and platforms.

   The tool prototype REX will be developed in order to provide support for designing rules and transforming rules from REX to timed-automata and from REX to different platforms for execution. The process of automating analysis and transformation will be performed in the following steps:

   - Firstly, automatic transformation will be realized for rules with primitive events and simple (true or false) conditions with immediate coupling mode and set oriented transition granularity.

- Secondly, automatic transformation will be realized for rules with composite events. Operators identified in Paton and Diaz survey [29] will be supported in recent and chronicle consumption mode.

- Thirdly, automatic transformation will be realized for rules with time constraints, such as deadlines and expiration times on composite events.

- Optionally, create a powerful user interface for REX and implement ability to elaborate with different execution models e.g. different coupling modes and different transition granularity. Additionally, ability to show traces from the analysis in UPPAAL is an optional task.

The target platform considered in this objective will be Polyhedra[30] and the timed-automata analysis tool considered will be UPPAAL[10].

5. **Perform case study to validate approach**
   A case-study will be performed to validate the usefulness of our rule development approach. The set of rules to be used in the case study will be specified in REX and the expressiveness of the rule language will be limited by information retrieved from Polyhedra and UPPAAL.

   The set of rules will be transformed to UPPAAL for analysis before an executable set of rules are created for Polyhedra. The set of rules will be as realistic as possible to reveal the performance of our development approach and scalability of analyzing rules in UPPAAL. Ideally, an existing rule-based system are identified and used as case-study object.

7 Current status

In this section, a brief summary of current status for each of the objectives are given. The following sections gives an in depth presentation of some of the results.

Objective 1 The rule language of REX will reflect our work of modelling rules in timed automata. REX currently support composite events with the set of operators presented in [21], priorities on rules, immediate coupling mode and set oriented transition granularity. Additional modelling will be performed to extend the ability to express different variations of execution models in timed-automata to subsume a large set of execution models.

Objective 2 address development of novel XML-schema and transformation algorithms. An algorithm for transforming composite events to small

timed automata models (patterns), representing the behavior of composite events in different consumption modes are identified and described in [21] and a short summary is given in section 10 in this thesis proposal. Additionally, a formal proof verifying equality in behavior between a composite event and its timed automaton pattern is presented in the Appendix.

Objective 3 address the issue of performing model-checking on a set of rules transformed to timed automata. This work is not yet initiated.

Objective 4 address the issue of automating the process of transforming rules between different representations through tool support. A first feasibility study has been performed on automating the process of transforming a set of rules with primitive events, immediate coupling mode and set oriented transition granularity to an UPPAAL specification. Using UPPAAL, it is possible to perform simulation and model checking on the set of rules to, for example, reveal if the set of rules has non-terminating behavior and what effect knowledge of execution times have on the ability to analyze rule behavior. Moreover, support for analyzing composite events with expiration times automatically transformed to UPPAAL patterns are implemented in REX as described in section 9.

Objective 5 A case study will be performed validating the development approach, the case study is not yet initiated.

8 Overview of development approach

Figure 2 shows an overview of our development approach. The flow of information in the development process is shown as solid lines while dashed lines shows representation type of exchanged information. The representation of execution and knowledge model will include information about how rules are processed during runtime.

The rounded rectangle *REX* represent the forthcoming development tool, *Platform for rule execution* represent, for example, Polyhedra [30], ruleCore [33] or DeeDS [6] and *Timed automata Analysis tool* represent the analysis tool UPPAAL [10].

The flow of information proceeds as follows;

1. Information about the execution model of the current target platform in the forthcoming representation.
2. Rules designed in REX is transformed to a representation readable by UPPAAL as described in Section 10.
3. Feedback from UPPAAL
4. Rules designed in REX and analyzed in the previous steps are transformed to platform specific syntax for execution

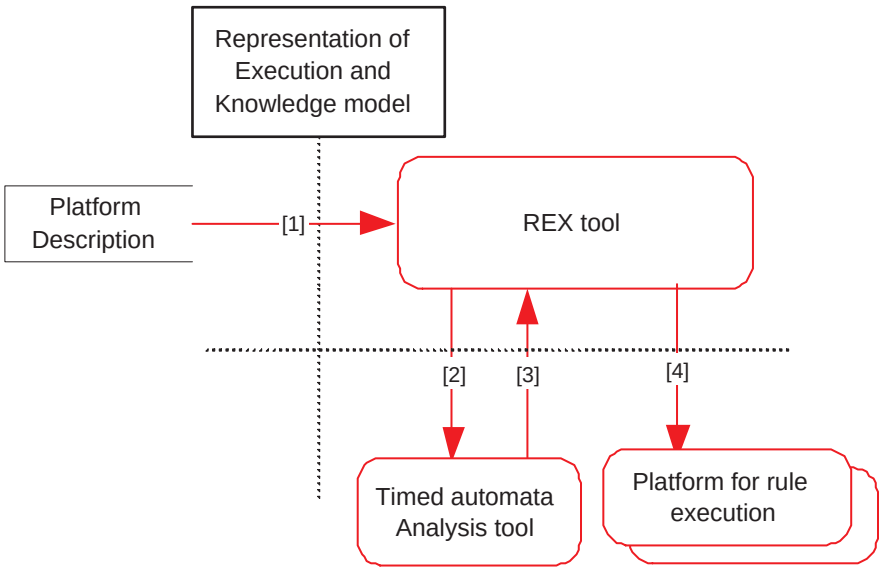

Figure 2: Overview of development approach.

9 Specifying events in REX

All events specified in REX, both composite and primitive, are viewed in a tree structure as shown to the left in Figure 3. Nodes with one piece of a puzzle are primitive events and nodes with two pieces connected are composite events. The properties of the current event are viewed and changed in the property table shown to the right in Figure 3.

A new event type is created by right clicking in the event tree. The properties of the event is specified in the new events properties table. Each event must have a unique name (the type of the event) and when a new event type is created, it is available to be chosen as an initiator or terminator for a composite event.

The current version of REX support transformations from REX to UPPAAL of primitive and composite events. The primitive events can be time events, triggered on a definite time or a time relative to some other occurrence, an external event (e.g sensor event) or internal event (e.g. update of a tuple in a database). The operators currently supported are Times(n,E) (triggered after n occurrences of event type E), sequence, conjunction, disjunction, non-occurrence in recent and chronicle context with or without expiration times and with or without parameters. The current version of REX only support non-interval based detection semantics, i.e. composite events are detected when their terminator occur.

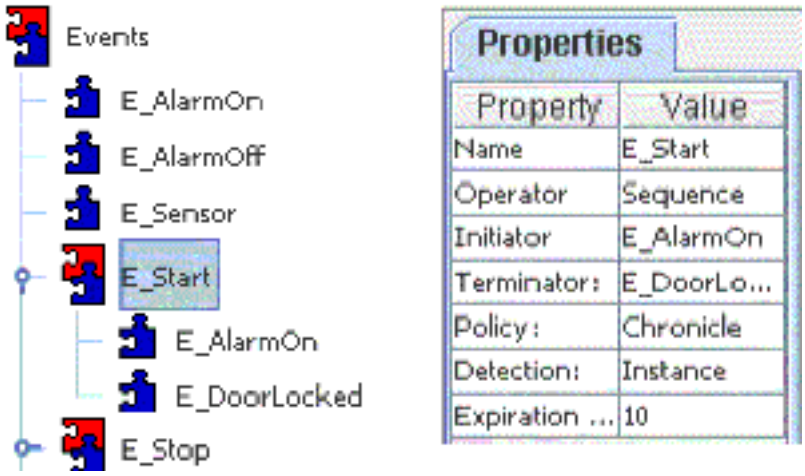

Figure 3: Representation of events in REX

10 Transformation approach

Since a timed automaton in UPPAAL is built up by a set of synchronizing timed automata, each simulating a process, it is possible to map each composite event to a timed-automaton process. An event *type* in the event specification is mapped to an instance of a channel in UPPAAL and an event *occurrence* is represented by synchronizing on the channel representing the event type.

Using this semantics, Automaton1 in Figure 1 signals that an event of type E_1 has occurred by taking the transition marked with channel $E_1!$ simultaneously as each automaton subscribing to events of type E_1 (in this example Automaton2), takes their corresponding transitions marked with $E1?$.

10.1 Transforming event parameters

Event parameters are sent with global messages and stored locally in each composite event automaton. In the example represented in Figure 1, the global variable *message* is sent by Automaton1 and received by Automaton2.

In chronicle consumption mode, a composite event is composed by the earliest contributing event occurrence that is not used by a previous instance of the composite event. This implies that parameters from occurred but not yet used event occurrences must be stored until they are used. In our approach, parameters are stored in a local array and a counter o_i is used to count number of occurred events of type E_i and a counter c_i is used to count number of consumed occurrences of type E_i. In chronicle context, an event is consumed when it is used or invalidated. An event is *used* when it has contributed to a composite event occurrence of the type modelled in the automaton and *invalidated* when the expiration time expires or if it, for example, is a terminator type and the composite event is not yet initiated.

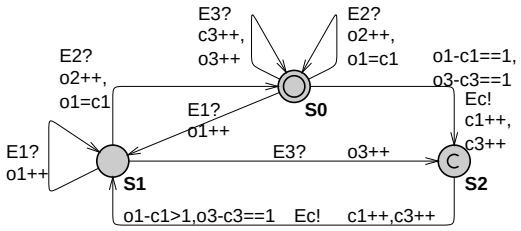

Figure 4: TOP for non-occurrence in chronicle context.

10.2 Timed-automata operator pattern (TOP)

Each event operator is transformed to a small timed-automaton operator pattern (TOP) [22]. Typically, an operator pattern starts in an initial state (marked *Initial* and with double circles) waiting for a contributing event occurrence. When at least one instance of each contributing event has occurred, the pattern reach state *Triggered* and a composite event is signaled by synchronizing on the channel representing the type of the composite event occurrence.

State *Triggered* is marked as committed, (C) implying that next transition, signaling the composite event occurrence, is immediately taken. Parameters attached to contributing events can be stored and assigned to parameters of the composite event.

10.3 Timed-automata operator pattern for $E_c = N(E_1, E_2, E_3)$

Figure 4 shows the timed automata pattern for non-occurrence $E_c = N(E_1, E_2, E_3)$ in chronicle context. An occurrence of event type E_c is generated when E_1 has occurred followed by an occurrence of type E_3 with no occurrence of type E_2 in between.

The automaton starts in location initial where it remains if an event of type E_2 or E_3 occurs before the composite event is initiated. If an event of type E_1 occurs, the automaton moves to location S1. In location S1, if an event of type E_2 occurs, the automaton invalidates all previous occurrences of type E_1 and takes the transition back to S0. If an event of type E_1 occurs, a new instance of E_c is initiated and the automaton remains in S1. If an event of type E_3 occurs, the automaton takes the transition to location S2 and immediately generates an event of type E_c. If there are unused occurrences of type E_1 in the event history ($o1 - c1 > 0$) the transition back to S1 is taken, else the transition to S0 is taken.

10.4 Correctness of transformation

The success of our approach of verifying rules in a timed automata representation of the rule set relies on the correctness of the transformation process. The

verification of the timed automata representation of the rule set loses its value if the behavior of the specified rules is altered during the transformation.

To verify behavioral equivalence between the non-occurrence event and its timed automata representation a formal proof is performed. The proof is presented in Appendix.

11 Expected contributions

The expected contribution of this project is an approach for enabling utilization of an existing formal analysis tool when rule based real-time systems are developed.

A model for how to describe the execution model of a rule language with time constraints and dynamically restrict and tailor the language with respect to a specific platform using a specific execution model will be developed. The model will reveal what information is needed to describe the behavior of a set of executing rules and how this information can be represented in an XML based model. The model will provide means for making development tools for rule based systems less platform dependent.

An XML-schema for describing rules with real-time constraints will be developed in order to support exchange of rules with real-time constraints. Algorithms for transforming a rule-base with maintained semantics to the representations of Uppaal and Polyhedra will be developed.

A high-level property specification language tailored for verifying rule based systems will be developed, increasing the knowledge of how to take advantage of the verification power of a timed automata model-checker when rule-based systems are developed.

Further, our work will enhance the tool support for rule-based systems, particulary rule-based systems with time constraints, through the development tool prototype REX. The following features will be included in REX:

- Support for graphically designing a set of rules.
- Support for retrieving information about execution models and dynamically change the rule language to be limited by the execution model of the current target platform.
- Automatic support for transforming the rule base to Uppaal
- Automatic support for transforming the rule base to platform specific rules
- Support for defining verification properties relevant for rule based systems.

Hence, the result of our work will provide developers with a prototype for a powerful, extendable tool for developing predictable rule-based systems for different platforms.

References

[1] A. Aiken, J. Hellerstein, and J. Widom, "Static analysis techniques for predicting the behavior of active database rules," *ACM Transactions on Database Systems*, vol. 20, pp. 3–41, 1995.

[2] J. J. Alferes, R. Amador, E. Behrends, M. Berndtsson, F. Bry, G. Dawelbait, A. Doms, M. Eckert, O. Fritzen, W. May, P. Patranjan, L. Royer, F. Schenk, and M. Schroeder, "Specification of a model, language and architecture for reactivity and evolution., Tech. Rep. REWERSE deliverable I5-D4, 2005, 2005.

[3] E. T. Almeida, J. E. Luntz, and D. M. Tilbury, "Modular Finite State Machines Implemented As Event-Condition-Action Systems," in *16th IFAC World Congress, Prague, Czech Republic, July 4-8 2005*, 2005.

[4] R. Alur, C. Courcoubetis, and D. L. Dill, "Model-checking for real-time systems," in *In 5th Symposium on Logic in Computer Science (LICS'90)*.

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

[6] S. F. Andler, J. Hansson, J. Eriksson, J. Mellin, M. Berndtsson, and B. Eftring, "Deeds towards a distributed active and real-time database system." vol. 25, pp. 38 – 40.

[7] J. Bailey, G. Dong, and K. Ramamohanarao, "Decidability and undecidability results for the termination problem of active database rules," in *PODS '98: Proceedings of the seventeenth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems*. New York, NY, USA: ACM Press, 1998, pp. 264–273.

[8] E. Baralis and J. Widom, "An algebraic approach to static analysis of active database rules," *ACM Transactions on Database Systems*, vol. 25, no. 3, pp. 269–332, 2000.

[9] G. Behrmann, A. David, and K. G. Larsen, "Tutorial on uppaal," in *In proceedings of the 4th International School on Formal Methods for the Design of Computer, Communication, and Software Systems (SFM-RT'04) LNCS 3185*, 2004.

[10] J. Bengtsson, K. G. Larsen, P. Pettersson, and W. Y., "Uppaal - a tool suite for automatic verification of real-time systems," *In Hybrid Systems III: Verification and Control*, pp. 232–243, 1996.

[11] M. Berndtsson, J. Mellin, and U. Högberg, "Visualization of the composite event detection process," in *International Workshop on User Interfaces to Data Intensive Systems (UIDIS), Edinburgh, 5th-6th September, IEEE Computer Society.*, 1999, pp. 118–127.

[12] J. P. Bowen and M. Hinchey, "Ten commands of formal methods," in *High-integriyt system specification and design*, J. P. Bowen and M. Hinchey, Eds. Springer-Verlag, 1998, pp. 215–230.

[13] A. P. Buchmann, J. Zimmermann, J. A. Blakeley, and D. L. Wells, "Building an Integrated Active OODBMS: Requirements, Architecture, and Design Decisions," in *Proceedings of the 11th International Conference on Data Engineering*. IEEE Computer Society Press, 1995, pp. 117–128.

[14] A. Buchmann, H. Branding, T. Kudrass, and J. Zimmermann, "Reach: A real-time, active and heterogeneous mediator system," *IEEE Quarterly Bulletin on Data Engineering, Special Issue on Active Databases*, vol. 15(1-4), pp. 44–47, 1992.

[15] H. C.A.R, "An overview of some formal methods of program design," *IEEE Computer*, vol. 32(1), pp. 85–91, 1987.

[16] S. Ceri and P. Fraternali, *Designing database applications with objects and rules*. Addison-Wesley, 1997.

[17] S. Chakravarthy, V. Krishnaprasad, E. Anwar, and S. Kim, "Composite events for active databases: Semantics, Contexts and Detection," in *Proceedings of VLDB*, 1994, pp. 606–617.

[18] S. Comai and L. Tanca, "Termination and confluence by rule prioritization," *IEEE Transactions on knowledge and data engineering*, vol. 15(2), pp. 257–270, 2003.

[19] O. Diaz, "Tool Support," in *Active Rules in Database Systems*, N. Paton, Ed., 1999, pp. 127–146.

[20] M. B. Dwyer, G. S. Avrunin, and J. C. Corbett, "Patterns in property specifications for finite-state verification," in *Software Engineering, 1999, Proceedings of the 1999 International Conference on*. IEEE, 1999, pp. 411–420.

[21] A. Ericsson and M. Berndtsson, "Detecting design errors in composite events for event triggered real-time systems using timed automata," in *First International Workshop on Event-driven Architecture, Processing and Systems (EDA-PS 06) Chicago*, 2006, pp. 39–47.

[22] A. Ericsson, R. Nilsson, and S. Andler, "Operator patterns for analysis of composite events in timed automata," in *WIP Proceedings : 24th IEEE Real-Time Systems Symposium, Cancun, Mexico.*, 2003.

[23] A. Jahne, S. D. Urban, and S. W. Dietrich, "Peard: A prototype environment for active rule debugging." *Journal of Intelligent Information Systems*, vol. 7, no. 2, pp. 111–128, 1996.

[24] G. Kappel, G. Kramler, and W. Retschitzegger, "TriGS debugger — A tool for debugging active database behavior," *Lecture Notes in Computer Science*, vol. 2113, pp. 410–421, 2001.

[25] H. Kopetz and P. Verissimo, "Real time and dependability concepts," in *Distributed systems*, 2nd ed., S. Mullender, Ed. Harlow: Addison-Wesley, 1993, pp. 411–446.

[26] W. May, J. J. Alferes, and R. Amador, "Active Rules in the Semantic Web: Dealing with Language Heterogeneity," in *Proceedings of International Conference on Rules and Rule Markup Languages for the Semantic Web, Galway, Ireland (10th–12th November 2005)*, ser. LNCS, vol. 3791, 2005, pp. 30–44.

[27] J. Mellin, "Resource predictable and efficient monitoring of events," Ph.D. dissertation, Linköping University, Sweden, 2004.

[28] N. W. Paton, J. Campin, A. A. A. Fernandes, and M. H. Williams, "Formal specification of active database functionality: A survey," in *Proceedings of the 2nd International Workshop on Rules in Database Systems*, T. Sellis, Ed., vol. 985. Springer, 1995, pp. 21–35.

[29] N. W. Paton and O. Diaz, "Active database systems," *ACM Comput. Surv.*, vol. 31, no. 1, pp. 63–103, 1999.

[30] Polyhedra, "http://www.polyhedra.com/," 2005. [Online]. Available: http://www.polyhedra.com/

[31] K. Ramamritham, R. M. Sivasankaran, J. A. Stankovic, D. F. Towsley, and M. Xiong, "Integrating temporal, real-time, and active databases," *SIGMOD Record*, vol. 25, no. 1, pp. 8–12, 1996.

[32] I. Ray and I. Ray, "Detecting termination of active database rules using symbolic model checking," *Lecture Notes in Computer Science*, vol. 2151, pp. 266–279, 2001.

[33] M. Seiriö and M. Berndtsson, "Design and Implementation of an ECA Rule Markup Language," in *Proceedings of the 4th International Conference on Rules and Rule Markup Languages for the Semantic Web (RuleML-2005)*, ser. LNCS, vol. 3791. Springer, 2005, pp. 98–112.

[34] E. Simon and A. Kotz-Dittrich, "Promises and realities of active database systems," in *Proc. of the 21st Conference on Very Large Databases, VLDB95 Zurich*, Zurich, Switzerland, 1995, pp. 642–653.

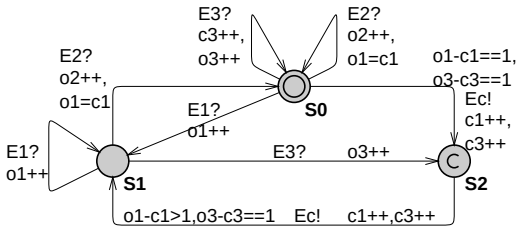

Figure 5: TOP for nonoccurrence in chronicle context.

T_i	E_i	**source**	**guard**	**action**	**target**
T_1	$E_1?$	s_0, s_1	true	$o_1 + +$	s_1
T_2	$E_2?$	s_0, s_1	true	$o_2 + +$, $c_1 := o_1$	s_0
T_3	$E_3?$	s_0	true	$o_3 + +$, $c_3 + +$	s_0
T_4	$E_3?$	s_1	true	$o_3 + +$	s_2
T_5	$E_c!$	s_2	$o_1 - c_1 = 1$ $o_3 - c_3 = 1$	$c_1 + +$, $c_3 + +$	s_0
T_6	$E_c!$	s_2	$o_1 - c_1 > 1$ $o_3 - c_3 = 1$	$c_1 + +$, $c_3 + +$	s_1

Table 1: TOP for nonoccurrence in chronicle context represented as a table.

A Appendix A

A.1 Example: TOP for Non-occurrence

In the following, the TOP for non-occurrence in chronicle context, shown in Figure 5, is used for exemplifying TOPs and a verification proof. We choose to show the non-occurrence operator since it includes both generation, usage and invalidation of event occurrences. The non-occurrence TOP represents the behavior of the composite event $E_c = N(E_1, E_2, E_3)$ which is generated if an occurrence of type E_3 is observed after an occurrence of E_1 with no observed occurrences of type E_2 in between.

Table1 represent the transitions of the TOP for $E_c = N(E_1, E_2, E_3)$ shown in Figure 5. The variables o_i and c_i are defined as counters for occurred and consumed event occurrences. The expression T_i refers to the transition represented in row i in Table 1. For each row, column E_i specifies the occurred event type, *source* specifies transition source location, *guard* specifies transition guard (true for no guard), *action* specifies the assignments performed during the transition and the *target* column specifies the transitions target location.

The TOP in Figure 5 starts in location s_0. An event occurrence of type E_1 (T_1) initiates the composite event and the TOP transfer to s_1. If an event of type

E_2 occur, all unconsumed occurrences of type E_1 (T_2) are invalidated, since they can no longer be used to form a composite event of type $E_c = N(E_1, E_2, E_3)$. If an occurrence of type E_3 is observed in s_0 (T_3), it is immediately invalidated, however, if it is observed in s_1 (T_4), an event of type E_c can be generated and the TOP transfer to s_2. In s_2, a composite event $E_c = N(E_1, E_2, E_3)$ is generated using the earliest unconsumed occurrences of type E_1 and E_2. If there is exactly one unconsumed occurrence of type E_1 in the event history, the TOP transfer to s_0 (T_5), else it transfer to s_1 (T_6).

A.2 Proof

The behavioral equivalence of the analyzed timed automaton and its corresponding rule base is essential for our development approach. To prove equivalent behavior between timed automata and rules triggered by a composite event, the formal event specification language Solicitor [27] is used as low-level target language for verifying the event part of the rule. Solicitor expressions can be converted to executable code used in a resource predictable event monitor [27]. In the following, a brief summary of Solicitor is given.

A.2.1 Basic definitions of formal event schema

The symbols used here are previously defined in Solicitor [27]. An event occurrence is defined by $\Gamma(E, [t, t'])$ that is read "E has occurred throughout $[t, t']$", and γ (where $\gamma = \Gamma(E, [t, t'])$ denotes generated event occurrences. The symbols i–l are used to index event types. The function $type(\gamma)$ returns the event type E of the event occurrence γ. For any γ_i, γ_j the relation *precedes* $\gamma_i \prec \gamma_j$ holds true if γ_i precedes γ_j, and the relation *overlap* $\gamma_i \parallel \gamma_j$ holds true if γ_i and γ_j overlaps. Finally, the predicate $gen(\gamma)$ holds true if there is a γ that has occurred. An event type E_i contributes to event type E_c iff E_c is a composite event type that is fully or partly composed by E_i. If, for example, $E_c = (E_1 \wedge E_2)$, then $contr(E_1, E_c)$ and $contr(E_2, E_c)$.

Definition 1 *A generated event occurrence is observed by a composite event if its event type contributes to the composite event type.*
$\forall Ec, E_i, \gamma_i :$

$$\frac{E_i = type(\gamma_i) \wedge gen(\gamma_i) \wedge contr(E_i, E_c)}{obs(E_c, E_i, \gamma_i)}$$

The $used(E_c, \gamma_i, \gamma_c)$ predicate is true when a composite event type E_c has used the event occurrence γ_i to generate the composite event occurrence γ_c. The $inv(E_c, \gamma_i)$ predicate is true when the composite event type E_c can not use the event occurrence γ_i to form a composite event occurrence. This is for example true for all unused occurrences of type E_1 if they are followed by an occurrence of type E_2.

Definition 2 *The consumed predicate* $cons(E_c, E_i, \gamma_i)$ *is true if the composite event type* E_c *has used or invalidated event occurrence* γ_i *of type* E_i.

$$cons(E_c, E_i, \gamma_i) \triangleq obs(E_c, E_i, \gamma_i) \rightarrow (inv(E_c, \gamma_i) \vee \exists \gamma_c(used(E_c, \gamma_i, \gamma_c)))$$

The parameters evaluated in the context predicate, defined by Mellin [27] and redefined in definition 3, are the currently evaluated composite event type (E_c), the current potential initiator occurrence (γ_α), the current potential terminator occurrence (γ_ω), the set of event types whose occurrences may be initiators ($\mathcal{E}_\alpha$) and the set of event types whose event occurrences may be terminators ($\mathcal{E}_\omega$).

Hence, in the proof example, (γ_α) is the earliest unconsumed occurrence of type E_1, (γ_ω) is the earliest unconsumed occurrence of type E_3, ($\mathcal{E}_\alpha$) is $\{E_1\}$ and ($\mathcal{E}_\omega$) is $\{E_3\}$.

Definition 3 *The chronicle context predicate [27] is true when the occurrence* γ_α *is the earliest available initiator occurrence and* γ_ω *is the earliest available terminator occurrence for the composite event* E_c *in chronicle context.*

$$\begin{aligned}
&chronicle(E_c, \gamma_\alpha, \gamma_\omega, \mathcal{E}_\alpha, \mathcal{E}_\omega) \triangleq \\
&\quad \exists E_\alpha(E_\alpha = type(\gamma_\alpha) \wedge obs(E_c, E_\alpha, \gamma_\alpha) \wedge E_\alpha \in \mathcal{E}_\alpha \wedge \\
&\qquad \neg cons(E_c, E_\alpha, \gamma_\alpha) \wedge \\
&\qquad \forall \gamma' \exists E'_\alpha(E'_\alpha = type(\gamma') \wedge E'_\alpha \in \mathcal{E}_\alpha \wedge \gamma' \neq \gamma \wedge \\
&\qquad\quad obs(E_c, E_\alpha, \gamma') \rightarrow (cons(E_c, E'_\alpha, \gamma') \vee \gamma_\alpha \prec \gamma'))) \wedge \\
&\quad \exists E_\omega(E_\omega = type(\gamma_\omega) \wedge obs(E_c, E_i, \gamma_\omega) \wedge E_\omega \in \mathcal{E}_\omega \wedge \\
&\qquad \neg cons(E_c, E_\omega, \gamma_\omega) \wedge \\
&\qquad \forall \gamma' \exists E'_\omega(type(\gamma') = E'_\omega \wedge E'_\omega \in \mathcal{E}_\omega \wedge \gamma' \neq \gamma \wedge \\
&\qquad\quad obs(E_c, E_i, \gamma') \rightarrow (cons(E_c, E'_\omega, \gamma') \vee \gamma_\omega \prec \gamma')))
\end{aligned}$$

A.2.2 Non-occurrence operator rule

The foundation of the event schema described by Mellin [27] are the *generation* and *non-generation rules* defining the event operator semantics. The generation rules define situations when composite events are generated. The non-generation rules are help rules that are used to define situations when event occurrences of contributing event types can not be part of any future composite event occurrences. A non-occurrence $E_c = N(E_1, E_2, E_3)$ is generated if an event of type E_1 occurs followed by an occurrence of type E_3 with no occurrence of type E_2 between them. In the following, the generation rule (n-rule) and the non-generation rule (n"-rule) for the non-occurrence operator are redefined from [27].

When the precondition for the n-rule is true in Solicitor, a new composite event of type E_c is generated. The contributing event occurrences are thereby used by the composite event type.

Definition 4 ***Non occurrence Rule (n-rule)***
Let $E_c = N(E_1, E_2, E_3)$ *and*

$\gamma_c \triangleq \Gamma(E_c, [start(span(\gamma_1)), end(span(\gamma_3))])$ *in the n-rule n:*

$\forall E_c, \gamma_1, \gamma_3 :$
$obs(E_c, E_1, \gamma_1) \wedge obs(E_c, E_3, \gamma_3)$
$chronicle(E_c, \gamma_1, \gamma_3, \{E_1\}, \{E_3\}) \wedge \gamma_1 \prec \gamma_3 \wedge \neg(\gamma_1 || \gamma_3)$
$\neg\exists\gamma_2((obs(E_c, E_2, \gamma_2) \wedge \gamma_1 \prec \gamma_2 \wedge \neg(\gamma_1 || \gamma_2) \wedge$
$\gamma_2 \prec \gamma_3 \wedge \neg(\gamma_2 || \gamma_3))$

$$\frac{}{used_\alpha(E_c, \gamma_1, \gamma_c) \quad gen(\gamma_c) \quad used_\omega(E_c, \gamma_3, \gamma_c)}\, n$$

When the precondition for the n"-rule is true, all unconsumed event occurrences of type E_1 are invalidated and will not contribute to any forthcoming composite event occurrences of type E_c. The predicate $chronicle_\alpha(E_c, \gamma_1, \{E_1\})$ defined in [27] is true for the earliest unconsumed occurrence of type E_1.

Definition 5 ***Non occurrence Rule (n"-rule)***
$\forall E_c, \gamma_1 :$
$obs(E_c, \gamma_1) \wedge type(\gamma_1) = E_1 \wedge chronicle_\alpha(E_c, \gamma_1, \{E_1\}) \wedge$
$\forall\gamma_3((obs(E_c, \gamma_3) \wedge type(\gamma_3) = E_3 \wedge$
$chronicle(E_c, \gamma_1, \gamma_3, \{E_1\}, \{E_3\}) \wedge$
$\gamma_1 \prec \gamma_3 \wedge \neg(\gamma_1 || \gamma_3)) \rightarrow$
$\exists\gamma_2(type(\gamma_2) = E_2 \wedge$
$\gamma_1 \prec \gamma_2 \wedge \neg(\gamma_1 || \gamma_2) \wedge \gamma_2 \prec \gamma_3 \wedge \neg(\gamma_2 || \gamma_3))$

$$\frac{}{inv_\alpha(E_c, \gamma_1)}\, n"$$

In short, the n-rule together with the n"-rule states that a composite event of type $E_c = N(E_1, E_2, E_3)$ is generated when there is an event occurrence of type E_1 and one event occurrence of type E_3 with no event occurrence of type E_2 in between. The chronicle context predicate ensures that the earliest available (not already used or invalidated) occurrences of type E_1 and E_3 contributes to E_c. Once an event occurrence has contributed to a composite event, it is marked as used and can not contribute to additional event occurrences of type E_c. Further, if an event of type E_2 occurs, all previously unconsumed occurrences of type E_1 are invalidated.

A.2.3 Timed automata to Solicitor

To be able to prove equivalence of semantics between timed automata operator patterns and Solicitor, rules for mapping constructs in timed automata to inference rules and predicates in Solicitor is defined. The composite event to be generated is $E_c = N(E_1, E_2, E_3)$ in chronicle context. Using the notation explained in section **??** and **??**, the following axiom holds:

Axiom 1
if $(s_m, t_m) \stackrel{E_i!}{\rightarrow} (s_n, t_n)$ *then* $\{\exists\gamma(gen(\gamma) \wedge E_i = type(\gamma))\}$

if $(s_m, t_m) \stackrel{E_i?}{\rightarrow} (s_n, t_n)$ *then* $\{\exists \gamma_i(obs(E_c, E_i, \gamma_i))\}$

Axiom 1 states that if the TOP sends on channel E_i, then an event of type E_i is generated and if the TOP receives on channel E_i, then an event of type E_i is observed by the composite event E_c representing the TOP.

Definition 6 *The number of events of type E_i that are observed by the composite event of type E_c is returned by the function obs♯*
$obs\sharp(E_c, E_i) = |\{\gamma | obs(E_c, E_i, \gamma)\}|$

Definition 7 *The number of events of type E_i that are consumed by the composite event of type E_c is returned by the function cons♯.*
$cons\sharp(E_c, E_i) = |\{\gamma | cons(E_c, E_i, \gamma)\}|$

Definition 8 *The occurrence index of the specific event occurrence γ_i of type E_i is returned by the function*
$obsIndex(E_c, E_i, \gamma_i)$.

$$obsIndex(E_c, E_i, \gamma_i) = |\{\gamma_i' | obs(E_c, E_i, \gamma_i') \rightarrow \gamma_i \prec \gamma_i')\}| + 1$$

Lemma 1 $\forall i(obs\sharp(E_c, E_i) = o_i)$

Proof: According to Axiom 1, an event occurrence of type E_i is observed if the TOP synchronizes on channel $E_i?$ and according to Table 1, the counter o_i is increased each time the TOP synchronizes on $E_i?$. □

Lemma 2 $\forall i(cons\sharp(E_c, E_i) = c_i)$

Proof: According to Table 1, c_i is increased when the semantics of the TOP is to invalidate or use an event of type E_i. □

Lemma 3 *The earliest unconsumed occurrence of type E_i has index $c_i + 1$.*□

Proof: Chronicle context is considered, implying that events occur and are consumed in chronicle order. The number of consumed occurrences are c_i and since all events are stored in occurrence order, the earliest unconsumed occurrence of type E_i has index $c_i + 1$. □

A.2.4 Mapping transitions to Solicitor

To be able to show equal behavior between the TOP for non-occurrence and $E_c = N(E_1, E_2, E_3)$ defined in Solicitor, the effect of each transition in Table 1 must be mapped to predicates in Solicitor.

Lemma 4 *Transition T_2 is mapped to*
$\exists \gamma_2(obs(E_c, E_1, \gamma_2)) \wedge \forall \gamma_1(cons(E_c, E_1, \gamma_1))$

Proof: According to Table 1 and Lemma 1, $\exists\gamma_2(obs(E_c, E_2, \gamma_2))$ is true since o_2 is increased in T_2. According to the n"-rule define in Definition 5, all unconsumed occurrences of type E_1 preceding an event occurrence of type E_2 are invalidated. Hence, after T_2, $\forall\gamma_1(cons(E_c, E_1, \gamma_1))\}$ holds, since all events occur in chronicle order and all events of type E_1 observed so far precedes the event of type E_2 observed by T_2. □

Lemma 5 *Transition T_1 is mapped to*
$\exists\gamma_1(obs(E_c, E_1, \gamma_1) \wedge obsIndex(E_c, E_1, \gamma_1) = o_1)$

Proof: According to Table 1, o_1 is increased during T_1, according to Lemma 6,
obsIndex returns the number of observed events of type E_1 preceding $\gamma_1 + 1$. Since o_1 is increased during T_1 and γ_1 is the latest occurrence
$obsIndex(E_c, E_1, \gamma_1) = o_1$ holds after T_1. □

Lemma 6 *Transition T_3 is mapped to*
$\exists\gamma_3(obs(E_c, E_3, \gamma_3) \wedge cons(E_c, E_3, \gamma_3) \wedge$
$obsIndex(E_c, E_3, \gamma_3) = o_3)$.

Proof: According to Table 1 and Lemma 1 and 2, an occurrence of type E_3 is both observed and consumed during T_3, hence, no unconsumed event occurrence of type E_3 are added to the history of events. □

Lemma 7 *Transition T_4 is mapped to*
$\exists\gamma_3(obs(E_c, E_3, \gamma_3) \wedge \neg cons(E_c, E_3, \gamma_3) \wedge$
$obsIndex(E_c, E_3, \gamma_3) = o_3)$.

Proof: According to Table 1 and Lemma 1, an event occurrence of type E_3 is observed during T_3. □

Lemma 8 *Transition T_5 and T_6 is mapped to*
$\exists\gamma, \gamma_1, \gamma_3(gen(\gamma) \wedge E_c = type(\gamma) \wedge cons(E_c, E_1, \gamma_1) \wedge$
$cons(E_c, E_3, \gamma_3) \wedge obsIndex(E_c, E_1, \gamma_1) = c_1 \wedge$
$obsIndex(E_c, E_3, \gamma_3) = c_3)$.

Proof: According to Table 1, Definition 1 and Lemma 2, an occurrence of type E_c is generated and one occurrence of type E_1 and E_3 are consumed during transition T_5 and T_6. Events are consumed in chronicle order and according to Lemma 3, $\forall i(c_i + 1)$ is the index of the earliest unconsumed occurrence of type E_i. During T_5 and T_6, the earliest unconsumed occurrence of type E_1 and E_3 are consumed and c_1 and c_3 are increased with one. □

A.2.5 Maintaining location invariants

The semantics for generating a composite event of type $E_c = N(E_1, E_2, E_3)$ in the TOP is to synchronize on channel E_c when the TOP is in location s_2 (since s_2 is source location for all transitions synchronizing on E_c!). To verify equality in behavior between the TOP and Solicitor it must be shown that the precondition for the n-rule in Solicitor is true exactly when the TOP is in location s_2.

In the following, location invariants stating what must be true in each location in the TOP is defined.

Definition 9 *In s_0 location invariant I_0 is true.*
$I_0 : \{\forall\gamma_1(cons(E_c, E_1, \gamma_1)) \wedge \forall\gamma_3(cons(E_c, E_3, \gamma_3))\}$

Definition 10 *In s_1 location invariant I_1 is true.*
$I_1 : \{\exists\gamma_1(obs(E_c, E_1, \gamma_1) \wedge \neg cons(E_c, E_1, \gamma_1)\wedge$
$\neg\exists\gamma_3(obs(E_c, E_3, \gamma_3) \wedge \neg cons(E_c, E_3, \gamma_3))\wedge$
$\neg\exists\gamma_2(obs(E_c, E_2, \gamma_2) \wedge \gamma_1 \prec \gamma_2 \wedge \neg(\gamma_1||\gamma_2)))\}$

Definition 11 *In s_2 location invariant I_2 is true.*
$I_2 : \{\exists\gamma_1(obs(E_c, E_1, \gamma_1) \wedge \neg cons(E_c, E_1, \gamma_1)\wedge$
$obsIndex(E_c, E_1, \gamma_1) = c_1 + 1\wedge$
$\exists\gamma_3(obs(E_c, E_3, \gamma_3) \wedge \neg cons(E_c, E_3, \gamma_3)\wedge$
$obsIndex(E_c, E_3, \gamma_3) = c_3 + 1 \wedge \gamma_1 \prec \gamma_3 \wedge \neg(\gamma_1||\gamma_3))\wedge$
$\neg\exists\gamma_2(obs(E_c, E_2, \gamma_2) \wedge \gamma_1 \prec \gamma_2 \wedge \neg(\gamma_1||\gamma_2) \wedge \gamma_2 \prec \gamma_3\wedge$
$\neg(\gamma_2||\gamma_3))\}$

The following lemmas show that for any permutation of event occurrences, I_0 defined in definition 9 always hold when the TOP is in location s_0, I_1 defined in definition 10 always hold when the TOP is in location s_1 and I_2 defined in definition 11 always hold when the TOP is in location s_2.

The proofs are performed by using a Hoare triple [15] for each transition presented in Table 1. The precondition P for a transition is the location invariant of the transitions source location, the command C is the mapping from transition action and synchronization to Solicitor and Q is the post-condition after the transition. If Q corresponds to the location invariant of the target location, then the transition is said to maintain the invariant of the target location.

Lemma 9 *Given that I_1 holds in location s_1 and I_2 holds in location s_2, I_0 holds as post condition for all transitions arriving to s_0.*

Proof: According to Table 1, s_0 is target location for transition T_2, T_3 and T_5, additionally, s_0 is the initial location for the TOP. In the initial state, no events have occurred and the proof is trivial. According to Table 1, T_2 starts in s_0 or s_1:

P: $I_0 \vee I_1$ (T_2 transitions can start either from s_0 or s_1.)
C: According to Lemma 4, T_2 is mapped to

$\exists\gamma_2(obs(E_c, E_1, \gamma_2)) \wedge \forall\gamma_1(cons(E_c, E_1, \gamma_1))$

According to Lemma 4, an occurrence of type E_2 invalidates all prior occurrences of type E_1. According to P, no unconsumed occurrences of type E_3 exists either, hence, I_0 holds after T_2.

Q: I_0

According to Table 1, T_3 starts in s_0.

P: I_0

C: According to Lemma 6 T_3 is mapped to

$\exists\gamma_3(obs(E_c, E_3, \gamma_3) \wedge cons(E_c, E_3, \gamma_3) \wedge$
$obsIndex(E_c, E_3, \gamma_3) = o_3)$.

The amount of unconsumed occurrences of type E_3 are maintained during T_3.

Q: I_0

According to Table 1, T_5 starts in s_2.

P: I_2

C: According to Lemma 8, T_5 is mapped to

$\exists\gamma, \gamma_1, \gamma_3(gen(\gamma) \wedge E_c = type(\gamma) \wedge cons(E_c, E_1, \gamma_1) \wedge$
$cons(E_c, E_3, \gamma_3) \wedge obsIndex(E_c, E_1, \gamma_1) = c_1 \wedge$
$obsIndex(E_c, E_3, \gamma_3) = c_3)$.

The guard $g_{2,0}$ on T_5 states $o_1 - c_1 = 1 \wedge o_3 - c_3 = 1$. According to Lemma 1 and 2, this implies that there is one unconsumed occurrence of type E_1 and one unconsumed occurrence of type E_3 in the history of events. According to Lemma 8, one occurrence of type E_1 and one occurrence of type E_3 are consumed during T_5 implying that there are no unconsumed occurrences of type E_1 or type E_3 in the event history after T_5 and hence, I_0 holds.

Q: I_0. □

Lemma 10 *Given that I_0 holds in location s_0 and I_2 holds in location s_2, I_1 holds as post condition for all transitions arriving to s_1.*

Proof: According to Table 1, s_1 is target location for transition T_1 and T_6.

P: $I_0 \vee I_1$ (The source location for T_1 is s_0 or s_1.)

C: According Lemma 5, T_1 is mapped to

$\exists\gamma_1(obs(E_c, E_1, \gamma_1) \wedge obsIndex(E_c, E_1, \gamma_1) = o_1)$

Starting with P, transition T_1 adds an occurrence of type E_1,hence I_1 is true after transition T_1.

Q: I_1

P: I_2 (The source location for T_6 is s_2.)

C: According to Lemma 8, T_6 is mapped to

$\exists\gamma, \gamma_1, \gamma_3(gen(\gamma) \wedge E_c = type(\gamma) \wedge cons(E_c, E_1, \gamma_1) \wedge$
$cons(E_c, E_3, \gamma_3) \wedge obsIndex(E_c, E_1, \gamma_1) = c_1 \wedge$
$obsIndex(E_c, E_3, \gamma_3) = c_3)$.

According to guard $g_{2,1}$ and Lemma 1 and 2 there are more than one unconsumed occurrence of type E_1 and one unconsumed occurrence of type E_3 in the event history if T_6 is taken. According to Lemma 8, one occurrence of type E_1 and E_3 are consumed during T_6, leaving one or more unconsumed occurrences of type E_1 and no unconsumed occurrences of type E_3 in the event history after T_6.

Q: I_1 □

Lemma 11 *Given that I_0 holds in location s_0 and I_1 holds in location s_1, I_2 holds as post condition for all transitions arriving to s_2.*

Proof: According to Table 1, s_2 is target location for T_4.

P: I_1 (The source location for transition T_4 is s_1.)

C: According to Lemma 7, T_4 is mapped to
$\exists\gamma_3(obs(E_c, E_3, \gamma_3) \wedge \neg cons(E_c, E_3, \gamma_3) \wedge$
$obsIndex(E_c, E_3, \gamma_3) = o_3)$.

Q: I_2 □

Lemma 12 *Location invariant I_2 defined in definition 11 is true exactly when the precondition for the n-rule redefined in definition 4 is true.*

Proof: We prove that I_2 is equivalent to the precondition of the n-rule for the contributing event occurrences γ_1 and γ_3

$I_2 : \exists\gamma_1(obs(E_c, E_1, \gamma_1) \wedge \neg cons(E_c, E_1, \gamma_1) \wedge$
$obsIndex(E_c, E_1, \gamma_1) = c_1 + 1 \wedge$
$\exists\gamma_3(obs(E_c, E_3, \gamma_3) \wedge \neg cons(E_c, E_3, \gamma_3) \wedge$
$obsIndex(E_c, E_3, \gamma_3) = c_3 + 1 \wedge \gamma_1 \prec \gamma_3 \wedge \neg(\gamma_1 || \gamma_3)) \wedge$
$\neg\exists\gamma_2(obs(E_c, E_2, \gamma_2) \wedge \gamma_1 \prec \gamma_2 \wedge \neg(\gamma_1 || \gamma_2) \wedge \gamma_2 \prec \gamma_3 \wedge$
$\neg(\gamma_2 || \gamma_3))$

According to Lemma 3 $cons\sharp_1(E_c, E_1, \gamma_1) + 1$ is the earliest unconsumed occurrence of type E_1 and $cons\sharp_3(E_c, E_3, \gamma_3) + 1$ is the earliest unconsumed occurrence of type E_3. This is exactly the definition of the predicate *chronicle* redefined in Definition 3, where γ_α is instantiated to γ_1, γ_ω is instantiated to γ_3, the set of initiator types is $\{E_1\}$ and the set of terminator types is $\{E_3\}$, hence, I_2 can be rewritten to:

$\exists\gamma_1, \gamma_3(obs(E_c, E_1, \gamma_1) \wedge$
$(obs(E_c, E_3, \gamma_3) \wedge chronicle(E, \gamma_1, \gamma_3, \{E_1\}, \{E_3\}) \wedge$
$\gamma_1 \prec \gamma_3 \wedge \neg(\gamma_1 || \gamma_3)) \wedge$
$\neg\exists\gamma_2(obs(E_c, E_2, \gamma_2) \wedge \gamma_1 \prec \gamma_2 \wedge \neg(\gamma_1 || \gamma_2) \wedge \gamma_2 \prec \gamma_3 \wedge \neg(\gamma_2 || \gamma_3))$

The predicate above is the precondition for the n-rule for the specific occurrences γ_1 and γ_3. □

Theorem 1 *The TOP for non-occurrence synchronize on channel E_c exactly when the preconditions for the n-rule is true in Solicitor.*

Proof: The TOP synchronizes on channel E_c in location s_2. According to Lemma 9, 10 and 11, I_2 is true exactly when the TOP is in location s_2 and nowhere else. According to Lemma 12, I_2 is true exactly when the preconditions for the n-rule is true in Solicitor, implying that the TOP synchronizes on channel E_c exactly when a composite event of type $E_c = N(E_1, E_2, E_3)$ is generated in Solicitor, hence, the TOP for non-occurrence has equal external behavior as the $E_c = N(E_1, E_2, E_3)$ for chronicle context specified in Solicitor. □

By Theorem 1 we can conclude that it is safe to transform the composite event $E_c = N(E_1, E_2, E_3)$ to an instantiated TOP for non occurrence and use it for analysis purposes.