Model-driven rule composition for event-based systems

Hannes Obweger*, Josef Schiefer, Martin Suntinger and Peter Kepplinger

UC4 Senactive,
Prinz-Eugen-Straße 72/1/5,
1040 Vienna, Austria
E-mail: hannes.obweger@uc4.com
E-mail: josef.schiefer@uc4.com
E-mail: martin.suntinger@uc4.com
E-mail: peter.kepplinger@uc4.com
*Corresponding author

Abstract: This article presents a novel framework for creating sense-and-respond rules, which allow detecting noteworthy event situations from streams of business incidents and responding to them in near real-time. Focusing on expressiveness as well as manageability, the proposed framework uses a model-driven approach for the rule definition, where the different aspects of a rule are specified in clearly separated, comprehensible sub-models. This includes models for event-type and correlation information, virtual business-object representations, event patterns ('sense') and actions ('respond'), as well as event processing networks. Event patterns are modelled in a visual decision graph from easy-to-understand pieces of pattern-detection logic, and/or from sub-level event patterns. The proposed system has been fully implemented with a service-oriented architecture. The rule model is illustrated with a business case from the workload-automation domain.

Keywords: rule composition; rule management; complex event processing; CEP.

Reference to this paper should be made as follows: Obweger, H., Schiefer, J., Suntinger, M. and Kepplinger, P. (2011) 'Model-driven rule composition for event-based systems', *Int. J. Business Process Integration and Management*, Vol. 5, No. 4, pp.344–357.

Biographical notes: Hannes Obweger received his Masters in Computer Science from the Vienna University of Technology and is presently pursuing his PhD, ibid. His work focuses on rule management for complex event processing systems. He is with UC4 Senactive, a Vienna-based company offering software for real-time sense-and-respond solutions.

Josef Schiefer received his PhD in Information Systems from the University of Vienna. He was a Researcher at IBM's Thomas J. Watson Research Center and one of the founders of Senactive Inc. Today, he is the Vice President for Development at UC4 Senactive.

Martin Suntinger received his Masters in Computer Science from the Vienna University of Technology and is currently working towards an MBA at the WU Executive Academy of the Vienna University of Economics and Business. His research interests include complex event processing and business intelligence, as well as innovation and new products management. He is with UC4 Senactive.

Peter Kepplinger is completing his Masters in Mathematics at the University of Vienna and especially interested in mathematical modelling and computational methods in population genetics. He is with UC4 Senactive.

1 Introduction

Complex event processing (CEP) enables real-time monitoring of business situations and automated decision-making in order to respond to threats or seize time-critical business opportunities. Applications thereof are manifold, ranging from logistics to fraud detection and automated trading; recently, the combination of CEP and business process management has led to the discipline of *event-driven business process management, ED-BPM* (von Ammon et al., 2008). The underlying business model is *sense-and-respond* (S&R) as proposed by Haeckel (1999). It is rooted in the idea that purposeful adaptive system design is more effective to deal with discontinuities and fast-moving industry environments as compared to traditional plan-and-execute strategies.

Event-based systems typically encompass a generic data integration layer with a multitude of adapters in order to

Copyright © 2011 Inderscience Enterprises Ltd.

receive (sense) business events from various source systems and respond back to these systems. Yet, the determinant of value and effectiveness of a CEP system is the evaluation process in between sensing and responding, namely the decision-making. Event-based applications typically use reactive event-pattern rules for modelling use-case specific decision logic. Event-pattern rules – which Luckham (2005) called 'the foundation for applications of CEP' – may be considered as event-processing logic in the form 'if situation *x* occurs in the event stream, then generate response *y*'. In the workload automation domain, for instance, an exemplary rule could be defined as follows: "If the frequency of error messages increases by a factor of two and tasks are delayed by more than 5%, then allocate additional resources for the execution environment".

For building large enterprise solutions based on CEP, mainly two requirements are key success factors: expressiveness and manageability. Clearly, expressiveness guarantees that a broad variety of use cases and scenarios can be modelled with the framework. Practical experience shows, however, that the development of event-based applications is a highly challenging task that requires design decisions at different levels of abstraction: Which real-world actions and state-changes shall emerge into event data? Which relationships exist between events? Which kind of event situation shall trigger what kind of response? Thus, equally important, manageability refers to ease of creation, administration and modification of complex rule sets.

In this article, we introduce the rule management of the event-based system *sense-and-respond infrastructure* (SARI) as originally proposed by Schiefer and Seufert (2005). In SARI, business situations and exceptions are modelled with *S&R rules* which have been designed to be created and modified by business users. SARI offers a user-friendly modelling interface for event-triggered rules with a correlation model and a graph for representing business situations as a combination of easy-to-understand pieces of pattern-detection logic. High expressiveness and usability is achieved by a rich set of predefined rule building blocks and a tailored expression language for formulating conditions and calculations. A clean separation of concerns, splitting the overall definition of a SARI application into a set of decoupled sub-models, facilitates manageability and reuse of components.

The remainder of this article is organised as follows: Section 2 discusses related work. In Section 3, we provide an overview of the proposed application model and the relationships between the various sub-models. SARI's event model, correlation model and business-entity model are introduced in Section 4 to Section 6. Section 7 presents a detailed description of SARI's rule model. The implementation of the event-processing model is discussed in Section 8. Section 9 shows SARI 'in action' with a real-world use case from the workload automation domain. Section 10 presents the results of an experimental evaluation. Section 11 concludes this article and gives an outlook to future work.

2 Related work

Related work can be divided into work on active event processing, event algebras in the active database community, work on event/action logics, updates, state processing/transitions, and temporal reasoning in the knowledge representation domain.

There has been a lot of research and development concerning knowledge updates and active rules in the area of active databases and several techniques based on syntactic [e.g., triggering graphs or activation graphs (Baralis and Widom, 1994)] and semantics analysis (e.g., Bailey et al., 1997) of rules have been proposed to ensure termination of active rules (no cycles between rules) and confluence of update programmes (always one unique outcome). The combination of deductive and active rules has also been investigated in different approaches mainly based on the simulation of active rules by means of deductive rules (Ludascher, 1998). However, in contrast to our work, these approaches often assume a very simplified operational model for active rules without complex events and event/condition/action (ECA) related event processing. ECA rules generally associate a triggering – possibly composite – event with one or more conditions and a set of actions. When the triggering event is detected and all conditions evaluate to true, the action part is executed.

Several CEP and event stream processing (ESP) systems have been developed, where many of them use an SQL-based approach for querying event streams. An example is Esper (http://esper.sourceforge.net), which is an open source event-stream engine that allows analysis of event streams with SQL-queries for defining correlations between events and for detecting event patterns. Aurora (Abadi et al., 2003), as well as its successors Borealis (Abadi et al., 2005) and Medusa (Zdonik et al., 2002), are also SQL-based processing engines, which provide efficient scheduling service and quality-of-service delivery mechanisms.

ruleCore (Seiriö and Berndtsson, 2005) is an event-driven rule processing engine supporting ECA rules, and providing a user interface for building composite events and rules.

Wu et al. (1998) propose an event correlation approach with rules in the 'conclusion if condition' form, which are used to match incoming events via an inference engine. Based on the results of each test and the combination of events in the system, the rule engine analyses data until it reaches a final state.

Chen et al. (2006) show an approach for rule-based event correlation. In their approach, they correlate and adapt complex/structural extensible markup language (XML) events corresponding to an XML schema. They describe an approach for translating hierarchical structured events into an event model that uses name-value pairs for storing data.

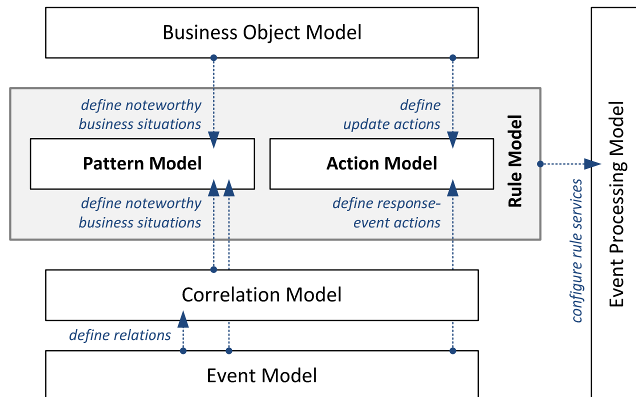
ECA rules have also been proposed by several authors for workflow execution (e.g., Barbará et al., 1994; Bussler and Jablonski, 1994; Dayal et al., 1990; Geppert and Tombros, 1998). In event-driven workflow execution, events and event-condition-action rules are the fundamental

mechanisms for defining and enforcing workflow logic. Processing entities enact workflows by reacting to and generating new events. The foundation for events facilitates the integration of processing entities into coherent systems. Some of these systems (Barbará et al., 1994), use composite events to detect complex workflow situations. EVE (Geppert and Tombros, 1998) is a system using ECA rules for workflow management addressing the problem of distributed event-based workflow execution.

3 A model-driven approach to event-based decision-making

In practice, manageability is as important for the success of rule-based event processing as is expressiveness. In order to provide manageability and usability also for large-scale solutions, SARI splits the overall definition of an event-based application into a collection of smaller, decoupled sub-models. Each sub-model thereby describes a certain aspect of an event application, beginning with the structure of all possible event data and ending with the orchestration of self-contained event-processing units such as adaptors and event services. Figure 1 shows the various sub-models of a SARI application along with the relationships between them. Detailed discussions of these models are presented in Section 4 to Section 8.

Figure 1 SARI application model (see online version for colours)



3.1 Event model

The event model provides abstract descriptions of all kinds of events that may occur within a SARI application, i.e., may emerge directly from the source system or be created virtually during the event processing. These descriptions – referred to as *event types* in the remainder of this article – declare all relevant characteristics of both the incident itself and the context in which it occurs. In the logistics domain, for instance, an event model would typically define events such as 'order placed', with properties such as the corresponding user account and the kinds and quantities of goods, 'order shipped', etc.

Event types form the foundation of any SARI application and generally allow higher-level models to be defined in a type-safe manner. The exact uses of event types in the various sub-models are discussed below.

3.2 Correlation model

The correlation model defines in an abstract manner whether two events *relate* to each other, i.e., whether they belong to a coherent sequence of real-world business incidents such as a business process. For instance, given an event model with two event types 'order placed' and 'shipment started', a correlation relationship 'order process' may link pairs of 'order placed' and 'shipment started' events by their order IDs. At runtime, so-defined classes of event situations are then used for partitioning the overall set of events and handling these partitions separately within the SARI application's event-processing logic. It is essential to note, however, that a correlation model does not define restrictions on the exact characteristics of a concrete event situation, such as specific event-attribute values or the ordering or quantity of events. SARI instead allows for a strict decoupling between the correlation aspect and the pattern-modelling aspect, thereby simplifying the definition of both halves and facilitating the reuse of correlation information across a SARI application.

The correlation model directly builds upon the *event model* for defining relationships in an abstract manner. By itself, the correlation model serves as a basis for the pattern-definition part of the *rule model*, where application designers may define classes of 'noteworthy' event situations by imposing additional constraints on sets of correlated events.

3.3 Business object model

The business object model enables application developers to define virtual representations of the various business objects existing in the underlying business environment. These entities may then be used to encapsulate certain kinds of business states in a controlled, intuitive and computationally efficient manner, and can be updated and queried from higher-level event-processing logic. Slimming down rule-definition logic by the handling of complex data over time, the business-object model thereby simplifies the definition and detection of noteworthy business situations.

Business objects are updated and queried through event-pattern rules as defined in the *rule model*. Albeit updates are always triggered by the occurrence of a respective event pattern, the definition of business objects themselves is generally independent from the event model, the correlation mode and the rule model.

3.4 Rule model

The rule model may be considered the key part of the proposed architecture and forms the basis for any kind of rule-based event processing in SARI. In so-called *S&R rules*, classes of noteworthy event situations – so-called *event patterns* – are associated with appropriate reaction logic, so-called *actions*. Whenever an incoming stream of

events matches the event pattern, the associated actions are triggered.

An event pattern may be considered an additional constraint on either a class of single business incidents as defined in the *event model* or on a class of business situations as defined in the *correlation model*. For instance, an event pattern ‘order delayed by x days’ would select from the overall set of all ‘order processes’ only those cases that are delayed by x days or longer. Actions allow generating response events as defined in the event model or updating virtual business object representations as defined in the *business object model*. S&R rules are referenced in the *event processing model* to be executed as part of a specific processing path through a SARI application.

3.5 Event processing model

The orchestration of self-contained event-processing agents, as well as their integration with underlying source systems, is finally described in an application’s event processing model. In so-called *event-processing maps*, the model describes:

a how real-world business occurrences are translated to events of respective event types

b how said events are processed in an orchestration of event services

c how response events trigger concrete actions in the underlying business environment.

S&R rules as defined in the underlying *rule model* are mapped to *rule services*, special event-processing units that evaluate sets of rules on the incoming stream of events and publish possible response events.

4 Event model

Forming the bottom layer of the proposed architecture, the event model provides abstract descriptions of all kinds of events that may emerge from the source system or be created virtually during the event processing.

SARI builds upon a strongly-typed event model that is oriented towards the type systems of modern object-oriented programming languages. Figure 2 sketches the meta-model for a SARI application’s event-type library. An event type $T = \{a_1, a_2, \ldots, a_n \mid a_i = (i_i, t_i)\}$ is defined by a set of *event* attributes, each having an identifier i and an event-attribute type. SARI supports three kinds of event-attribute types:[1]

- *Single-value types* include primitive types (such as integers, strings, etc.) as well as event types (i.e., an event may hold auxiliary events as event attributes).
- *Collection types* are lists of attribute-typed elements.
- *Dictionary types* eventually associate attribute-typed values with primitive-typed keys.

An event type T' may furthermore be in a *subtype relationship* with a base event type T; as usual, a sub event-type inherits all event attributes from the base type, i.e., $T \subseteq T'$. By definition, each event type must originate from a root event type ‘base event’. ‘Base event’ defines a timestamp ‘creation time’ – holding an event’s time of occurrence – as well as a unique identifier ‘ID’. For further details on SARI’s event model, the interested reader may refer to Rozsnyai et al. (2007). An exemplary event type ‘order received’ is depicted in Figure 3.

Figure 2 Event type meta-model

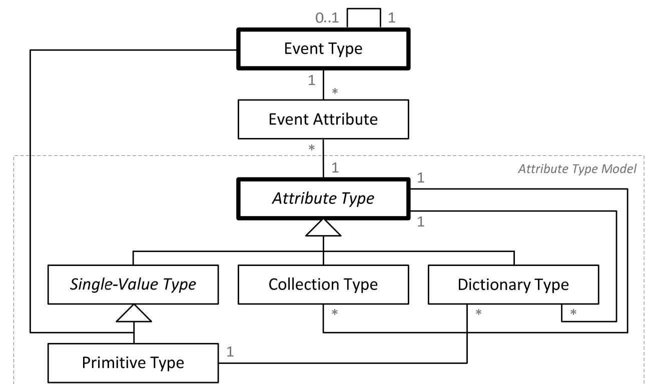
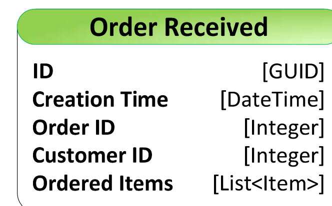


Figure 3 Exemplary event type (see online version for colours)



5 Correlation model

Setting up on the *event model*, the correlation model defines how instances of the various event types relate to each other in coherent sequences of incidents such as business processes. So-defined classes of event situations then allow partitioning the overall set of events during the event processing and the ex-post analysis of event data (Suntinger et al., 2008), and also form the basis for the *rule model* as described in Section 7.

SARI applications define correlation information in so-called *correlation sets* (Schiefer et al., 2009), a declarative model that allows incorporating and combining diverse correlation approaches through *correlation bands*. Each correlation set then corresponds to one class of event situations. In the logistics domain, for instance, a correlation set ‘shipment’ might correlate the events of all shipment processes as emerging from the source system. A concrete event-situation instance – e.g., the events of the specific shipment process #42 – is referred to as *correlation session*.

Figure 4 sketches the meta-model for correlation sets. A correlation set $s = \{b_1, b_2, \ldots, b_n\}$ is defined by a non-empty collection of correlation bands. Each correlation band describes a specific correlation approach for events of one or more event types as defined in the SARI application’s

event model, thereby defining a part of the overall event situation. A first correlation band b_i may, for instance, describe the correlation approach for 'order received', 'shipment ready' and 'order shipped' events as being based upon equal order IDs, while a second correlation band b_i may describe the correlation approach for 'order shipped' and 'shipment received' events as being based upon an explicit reference from the 'shipment received' events to the causally preceding 'order shipped' event (Figure 5).

Figure 4 Correlation set meta-model

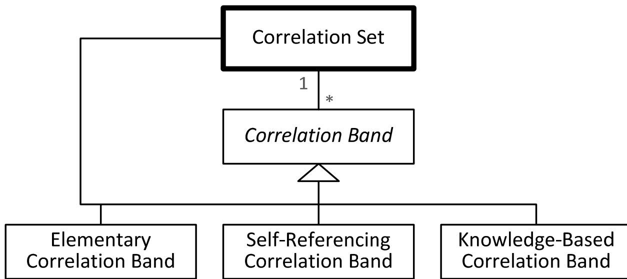
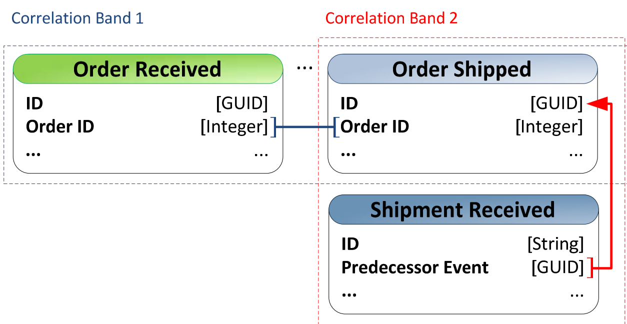


Figure 5 Exemplary correlation set (see online version for colours)



At the time of writing, SARI supports the following set of correlation bands:

- *Elementary correlation bands* correlate events of different types based upon equal event-attribute values. Let $\mathcal{T} = \{T\}$ be an event-type library. An elementary correlation band $e \subseteq \{(T, a) \mid T \in \mathcal{T}, a \in T\}$ is defined by a non-empty set of event types together with an event attribute per type. Given a correlation band $e = \{(T_1, a_1), (T_2, a_2), \ldots, (T_n, a_n)\}$, two events e_i: T_i and e_j: T_j are then correlated (and thus part of the same correlation session) if $value_{e_i}(a_i) = value_{e_j}(a_j)$. Note that a correlation band's event attributes do not necessarily have the same identifier. Also, note that a correlation set may comprise only one event type T, i.e., $n = 1$; then, it defines a subset of all T-events.
- *Self-referencing correlation bands* allow implementing scenarios where events explicitly refer to their (causal) predecessors. As with elementary correlation bands, a self-referencing correlation band $s \subseteq \{(T, a) \mid T \in \mathcal{T}, a \in T\}$ is defined by a non-empty set of event types, and, for each event type, an event attributes. Two events e and f, f of type T_i, are then correlated if $value_e(ID) = value_f(a_i)$, where 'ID' signifies the unique identifier attribute of an event.
- *Knowledge-based correlation bands* are similar to elementary correlation bands; however, for evaluating 'equality' between event-attribute values, an (external) knowledge base is queried. For instance, two string values 'Vienna' and 'Wien' could be detected as equivalent via an online dictionary. A knowledge-based correlation band $k = (e, b)$ therefore extends an elementary correlation set e by a knowledge-based b, offering methods for testing equality between two event-attribute values.
- *Correlation sets* may finally be (re-)used as correlation bands in higher-level correlation sets, which enables the hierarchical modelling of event situations.

6 Business object model

CEP systems define steering logic on event-based abstraction of real-world business environments. This approach fits particularly well for monitoring streams of self-contained business incidents; however, it tends to hit the wall when the overall state of real-world business objects needs to be derived from sequences of incremental updates. Consider an example from the system-monitoring domain, where a system administrator shall receive a notification whenever the number of alarms on a server exceeds a specified threshold. Here, a purely event-based system would have to correlate all alarm events within a sliding time window and perform a 'count' operation each time an alarm occurs. It is easy to see that for long time windows and/or high-frequent updates, this approach inevitably leads to serious performance issues.

SARI therefore provides a separate *business object model*, allowing application designers to encapsulate state in a controlled and intuitive manner. The business object model is implemented via so-called *business object providers*, plug-in-like components that – generally independent from other parts of the SARI architecture – manage specific kinds of business objects as application-wide, typed data-structures. Depending on their specific semantics, business object providers define a public interface for updating and querying the state of their data; read and write operations are accessible via SARI's rule model.

Given a certain business scenario, application designers will therefore:

a chose and incorporate appropriate business object providers

b for each business object provider, define the exact structure(s) of the required business object representations in the form of *business object types*.

In any case, a business object type specifies a (possibly composite) entity key; at runtime, entity keys then allow

identifying a specific instance of the given business object type.

SARI currently features two kinds of business objects:

1. *Measures* are the most basic kind of business object and basically may be considered numeric values that are set via a measure's update operations. Possible updates include basic operations such as increment/decrement, but also complex functions such as a moving average over a specified time window.

   A measure type $M = (K, v, h)$ is defined by:

   - A set of key properties $K = \{(i_1, t_1), (i_2, t_2), \ldots, (i_n, t_n)\}$; each key attribute (i, t) is defined by an identifier i and a data type t.
   - An initial value $v \in \mathbb{R}$.
   - A Boolean flag $h \in \{0, 1\}$, defining whether the system shall maintain the complete update history of a measure. The development of a measure over time can play a crucial role for the ex-post analysis of a system.

   Measures are typically used for counters. For implementing the above use-case of testing the number of server alarms against a defined threshold, an application designer would define a measure type 'alarms per server' with a single, string-typed key property 'server' and an initial value of zero. For each incoming alarm event, a rule r_1 would then increase the appropriate measure by one. A second rule r_2 would supervise the measure and, when it exceeds the threshold, notify the administrator.

2. *Entities* group sets of typed attributes and thus enable virtual representations of multi-variate real-world entities such as user accounts, suppliers, etc. Operations allow for continuously updating an entity's attributes, thereby keeping it in sync with its real-world correspondence. An entity type $e = (K, E)$ is defined by:

   - a set of key properties K
   - a set of entity attributes $E = \{(i_1, t_1, v_1), \ldots, (i_n, t_n, v_n)\}$; entity attributes define an identifier i_i, a type t_i and an initial value v_i: t_i.

7 Rule model

The detection of relevant patterns in continuous streams of business events is the key feature of CEP. Associating classes of noteworthy event situations with appropriate reaction logic, the rule model may therefore be considered the core of the proposed architecture. Together with the *event-processing model*, it defines the use-case specific decision logic of an event-based application.

To facilitate the reuse of event-processing logic across different business scenarios, SARI implements a strict decoupling between pattern modelling – i.e., the definition of noteworthy event situations – and action modelling, i.e., the definition of reaction logic. The respective sub-models, the *pattern model* and the *action model*, are discussed below. For the creation of full-fledged S&R rules from pattern-detection and reaction logic, SARI finally provides a two step workflow:

1. In a first step, IT experts and skilled business users create a catalogue of *pattern definitions* and *action definitions*. Pattern definitions describe classes of noteworthy event situations. Through a human-readable description of the described event situation and a set of input parameters, they enable business users to configure and apply the encapsulated pattern-detection logic without having to understand the event-based foundation of an application. Action definitions, similarly, abstract from concrete reaction logic.

2. In a second step, business users may instantiate concrete rule logic by assembling the required 'building blocks' and setting appropriate input-parameter values. A so-created S&R rule then encapsulates business logic in the form 'if situation occurs, then execute action(s)' and can be evaluated in *rule services* as defined in the event processing model.

7.1 Pattern model

Powerful pattern-detection logic is key to successful applications of CEP. However, especially for complex business processes comprising a large number of business incidents, describing classes of noteworthy event situations in an abstract manner may place heavy demands on users. SARI aims to simplify this process by employing a 'divide-and-conquer'-like approach to modelling pattern definitions, where application developers compose complex pattern-detection logic from easy-to-understand pieces of logic such as 'the occurrence of an event of type T, with certain attribute values' or 'the occurrence of a sub-pattern P'. These pieces – encapsulated in so-called *rule components* – are connected to each other in a directed, acyclic decision graph. At runtime, the predecessors in the graph are then considered as preconditions in the event-processing logic. To activate a component c – and thus bring it to play into the evaluation process – a concrete event situation must conform to (at least) one valid path through the decision graph. Depending on the evaluation result of c, further parts of the decision graph are activated, and so forth.

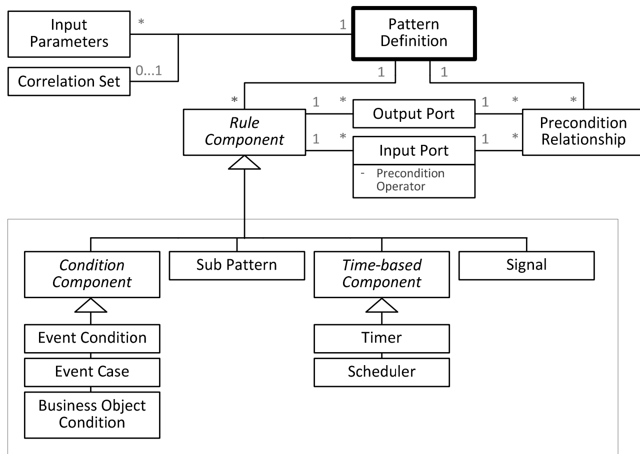
The described, graph-based structuring of pattern-detection logic suggests a graphical approach to pattern modelling, which may enable a comprehensive view of the overall pattern-detection logic as well as quick and easy access to single rule components. SARI provides a graphical pattern editor, which allows users to add, configure and connect graphical representations of rule components. Exemplary shapes are presented in Figure 8.

7.1.1 Meta model

Figure 6 shows the meta-model for pattern definitions. A pattern definition $p = (C, P, I, c)$ is defined by a set of rule

components *C*, a set of precondition relationships *P*, a set of input parameters *I*, as well as an optional correlation set *c*. A concrete realisation of the presented meta model is presented in Section 9.

Figure 6 Pattern definition model



Rule components

Encapsulating easy-to-understand pieces of pattern-detection logic, rule components may be considered the key element of any rule-based event processing in SARI. Depending on its implementation, a rule component $c \in C$ has a collection of input ports *IN* and a collection of output ports *OUT*; while input ports allow generally activating a rule component, output ports represent possible results of the encapsulated logic. Dependencies between components are modelled as precondition relationships between output ports and input ports. To allow multiple preconditions, a binary *precondition operator* specifies whether all (AND), at least one (OR) or exactly one (XOR) precondition must be fulfilled in order to activate an input port.

According to its specific role within a pattern definition, a rule component may furthermore define diverse *expressions* on:

a all business objects as defined in the business-object model

b correlation sessions as constituted by the pattern definition's correlation set *c*.

Whenever the rule component is triggered, these expressions are evaluated on the current values of the referred business objects and the events of the on-hand correlation session, respectively; for instance, if an evaluation is directly or indirectly caused by an incoming event *e*, the given expression is evaluated on the correlation session $S \ni e$ the event belongs to.[2]

Possible rule-component implementations are listed in the bottom of Figure 6. *Condition components*, the *sub-pattern component* and *time-based components* provide a powerful toolkit for describing classes of noteworthy event situations. For a detailed description of SARI's component library, the interested reader may refer to Figure 8. *Signals* are special components that notify the detection of an event situation to higher-level decision logic and are described in greater detail in the following section.

Precondition relationships

A precondition relationship $p = (in, out)$ associates an output port *out* of a rule component $r_i \in C$ with an input port *in* of another rule component $r_j \in C$. Cyclic dependencies are forbidden.

Input parameters

Input parameters of the form (i, t), where *i* is an identifier and *t* is a data type, allow adapting pattern-detecting logic to the concrete business scenario in which it is applied. When creating a pattern definition, an input parameter may be used as a typed placeholder across the various rule components of the decision graph. When using a concrete instance of the pattern definition – e.g., in a S&R rule – these placeholders are replaced by concrete values.

Correlation set

The proposed, model-driven approach to rule composition builds upon a strict decoupling of *event correlation* – defining classes of event situations on a common level, without further restrictions on the exact characteristics of a concrete situation instance – and *event-pattern modelling*, where for a given correlation set those characteristics of a concrete situation instance are defined that makes it *noteworthy* in a specific context. A pattern definition's correlation set consequently defines the class of event situations upon which a decision graph shall be evaluated; given a correlation set *s*, the decision graph is evaluated separately for each correlation session of *s*. When omitting the correlation configuration, a decision graph is evaluated independently for each incoming event.

Figure 7 Event correlation and pattern detection in SARI (see online version for colours)

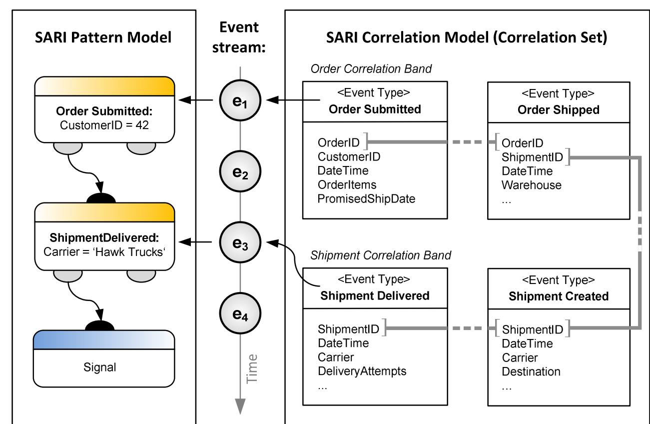
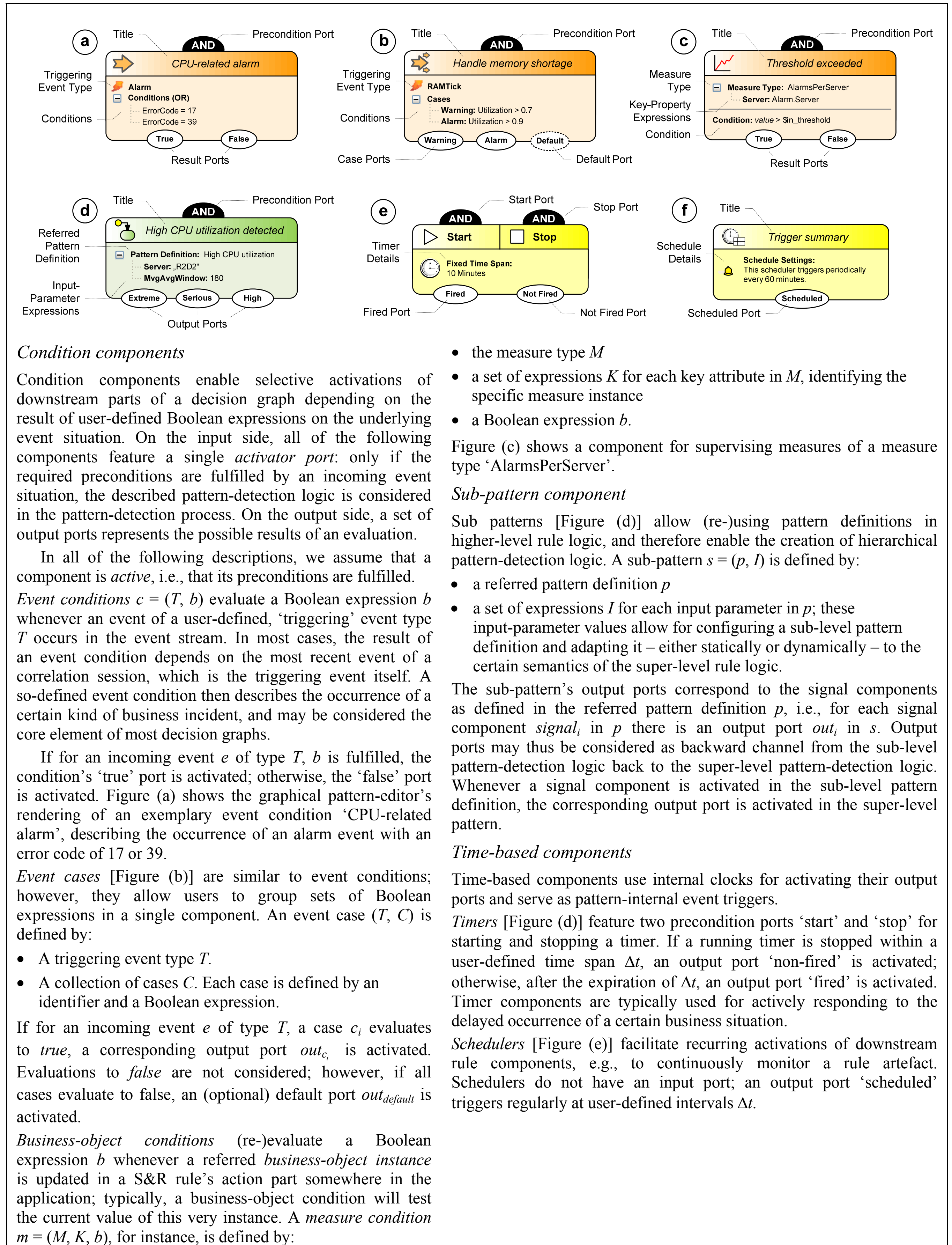


Figure 8 Rule components (see online version for colours)



Condition components

Condition components enable selective activations of downstream parts of a decision graph depending on the result of user-defined Boolean expressions on the underlying event situation. On the input side, all of the following components feature a single *activator port*: only if the required preconditions are fulfilled by an incoming event situation, the described pattern-detection logic is considered in the pattern-detection process. On the output side, a set of output ports represents the possible results of an evaluation.

In all of the following descriptions, we assume that a component is *active*, i.e., that its preconditions are fulfilled.

Event conditions $c = (T, b)$ evaluate a Boolean expression b whenever an event of a user-defined, 'triggering' event type T occurs in the event stream. In most cases, the result of an event condition depends on the most recent event of a correlation session, which is the triggering event itself. A so-defined event condition then describes the occurrence of a certain kind of business incident, and may be considered the core element of most decision graphs.

If for an incoming event e of type T, b is fulfilled, the condition's 'true' port is activated; otherwise, the 'false' port is activated. Figure (a) shows the graphical pattern-editor's rendering of an exemplary event condition 'CPU-related alarm', describing the occurrence of an alarm event with an error code of 17 or 39.

Event cases [Figure (b)] are similar to event conditions; however, they allow users to group sets of Boolean expressions in a single component. An event case (T, C) is defined by:

- A triggering event type T.
- A collection of cases C. Each case is defined by an identifier and a Boolean expression.

If for an incoming event e of type T, a case c_i evaluates to *true*, a corresponding output port out_{c_i} is activated. Evaluations to *false* are not considered; however, if all cases evaluate to false, an (optional) default port $out_{default}$ is activated.

Business-object conditions (re-)evaluate a Boolean expression b whenever a referred *business-object instance* is updated in a S&R rule's action part somewhere in the application; typically, a business-object condition will test the current value of this very instance. A *measure condition* $m = (M, K, b)$, for instance, is defined by:

- the measure type M
- a set of expressions K for each key attribute in M, identifying the specific measure instance
- a Boolean expression b.

Figure (c) shows a component for supervising measures of a measure type 'AlarmsPerServer'.

Sub-pattern component

Sub patterns [Figure (d)] allow (re-)using pattern definitions in higher-level rule logic, and therefore enable the creation of hierarchical pattern-detection logic. A sub-pattern $s = (p, I)$ is defined by:

- a referred pattern definition p
- a set of expressions I for each input parameter in p; these input-parameter values allow for configuring a sub-level pattern definition and adapting it – either statically or dynamically – to the certain semantics of the super-level rule logic.

The sub-pattern's output ports correspond to the signal components as defined in the referred pattern definition p, i.e., for each signal component $signal_i$ in p there is an output port out_i in s. Output ports may thus be considered as backward channel from the sub-level pattern-detection logic back to the super-level pattern-detection logic. Whenever a signal component is activated in the sub-level pattern definition, the corresponding output port is activated in the super-level pattern.

Time-based components

Time-based components use internal clocks for activating their output ports and serve as pattern-internal event triggers.

Timers [Figure (d)] feature two precondition ports 'start' and 'stop' for starting and stopping a timer. If a running timer is stopped within a user-defined time span Δt, an output port 'non-fired' is activated; otherwise, after the expiration of Δt, an output port 'fired' is activated. Timer components are typically used for actively responding to the delayed occurrence of a certain business situation.

Schedulers [Figure (e)] facilitate recurring activations of downstream rule components, e.g., to continuously monitor a rule artefact. Schedulers do not have an input port; an output port 'scheduled' triggers regularly at user-defined intervals Δt.

Figure 7 illustrates the proposed separation in a concrete example. On the left-hand side, the figure shows a decision graph for responding to all those orders of customer #42 that use a carrier 'Hawk Trucks Inc.' for the shipment. On the right-hand side, the figure shows the correlation set for associating related order and shipment events. During runtime, SARI uses the correlation set for correlating events before starting the actual rule evaluation. An incoming event e is then processed with the pattern-detection state for the certain correlation session $S \ni e$. Note that the decision graph in Figure 7 does not contain any information on how to link order events with related shipment events; this aspect is defined via the correlation set, which makes the pattern model less complex and focused on conditional logic only.

7.1.2 Signals

Signals are special rule components that, whenever activated, notify the detection of a noteworthy event situation to higher-level event-processing logic. In *super-level pattern definitions*, the signals of a pattern definition p are accessible via corresponding output ports of a *sub-pattern component* referring to p. In concrete S&R rules, *actions* are associated with the various signals of a rule's pattern definition; a S&R rule therefore encapsulates processing logic of the form 'If signal s is activated in the pattern-detection logic, then execute action.'

To provide access to selected characteristics of a matching event situation in a controlled and abstracted manner, signal components define a set of *output parameters O*, where

$$O = \left\{ out_1, out_2, ..., out_n \mid out_j = \left(i_j, t_j, e_j \right), 1 \le j \le n \right\}.$$

An output parameter $out = (i, t, e)$ is defined by an identifier i, a data type t and a correctly-typed expression e. When a signal component is activated, an output parameter's expression is evaluated on the underlying event data, thereby generating the output parameter's actual value.

Additionally, signals can be configured based on the following properties:

- *Reset rule state*: SARI's rule engine allows for completely resetting the state of a pattern definition for a given correlation session. If this property is set for an action component, the decision graph is reset automatically each time the component is activated. Newly incoming business incidents are then processed without considering previous events.
- *Silence interval*: Silence intervals allow an application designer to prevent *cascading actions*, e.g., when a certain threshold is exceeded. Beginning with the first successful execution of the described action, a so-configured component suppresses all further executions for the given period of time.

Taken together, signals along with their output parameters constitute the counterpart to above-described *input parameters* regarding the encapsulation and abstraction of concrete pattern-detection logic in pattern definitions.

7.2 Action model

The action model covers the 'respond' part of SARI's approach to rule-based event processing through so-called *action definitions*. Being associated with a signal of a S&R rule's pattern part, a concrete action is then executed each time the described event situation is detected in an incoming stream of events.

Actions currently supported by SARI are listed below. By definition, each action definition defines a set of input parameters I; as with pattern definitions, these input parameters allow configuring encapsulated reaction logic to the business scenario to which it is applied. When used as part of a concrete S&R rule, input-parameter expressions may comprise output parameters of the corresponding signal; the respond part of a rule may then be dynamically adapted to characteristics of the detected event situation.

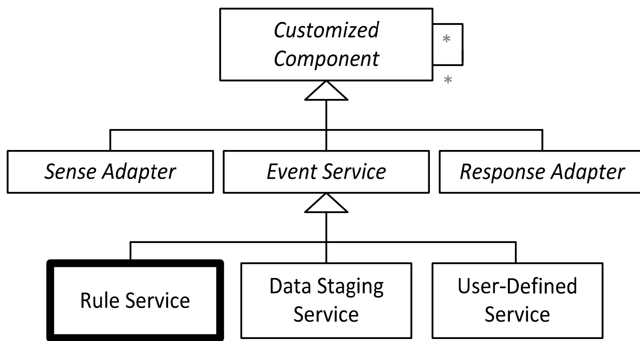
- Event actions describe a response event to be created and published whenever the action is executed. Such response may in turn trigger a concrete action in the source system (e.g., switch off a server) or an appropriate event service (e.g., send an e-mail), or serve as an input for another, downstream rule instance. An event-action definition $e = (T, E)$ is defined by:
  1. the event type $T = \{(i_1, t_1), (i_2, t_2), \ldots, (i_n, t_n)\}$ of the described response event
  2. a set of *attribute expressions* $E = \{e_i \mid e_i: I^* \rightarrow t_i, 1 < i < n\}$ for each event attribute $(i_i, t_i) \in T$, where I^* denotes the set of possible input-parameter assignments.

  Having the ability to directly or indirectly affect the source system, response events are the ultimate outcome of any event processing in SARI. Business-object actions as described below, by contrast, can be considered auxiliary actions for generating appropriate response events.
- Business-object actions comprise several actions for creating and updating *business objects*. Business-object actions are dynamically retrieved from plugged-in business object providers. For instance, a measure action $m = (K, v)$ for incrementing or decrementing instances of a user-defined measure type M is defined by:
  1. a set of key expressions $E = \{e_i \mid e_i: I^* \rightarrow t_i, 1 < i < n\}$ for each of n key properties (i_i, t_i) of M, identifying the concerned measure instance,
  2. an expression $v : I^* \rightarrow \mathbb{N}$ calculating the value of the *addend*, i.e., the value by which the measure shall be increased or decreased.

8 Event processing model

The proposed architecture is completed by the event processing model, which describes the overall event-processing logic of a SARI application in the form of so-called *event-processing maps*. Event processing maps are user-defined orchestrations of *sense adapters*, *event services*, and *response adapters*; the proposed meta-model is shown in Figure 9.

Figure 9 Event processing meta-model



Sense adapters translate real-world incidents into events of respective event types as defined in the application's event model. Typical sense-adapter implementations read data from message queues, log files, or are invoked actively by other parts of a company's IT landscape. Events are then routed through a network of event services according to user-specified links, so-called *event channels*, between the various map elements. In contrast to request/response-like interaction styles as, for instance, known from SOAP's message exchange patterns (W3C, 2007), event processing map use an asynchronous, message-based interaction style for routing events where a sender operates independently from the potential receivers of an event and their actions.

Event services receive input events from a collection of input ports, process these events based upon the requirements of the given business scenario, and publish response events to a collection of output ports. Event services cover any kind of event-triggered activity within an event-processing map; typical event-service implementations could, for instance, filter, transform or enrich event data. SARI offers a rich collection of configurable event services for the most common event-processing tasks. For custom tasks, a .NET API enables application developers to implement event services as .NET assemblies and incorporate arbitrary event-processing logic. *Rule services* are specialised event services that allow evaluating a set of *S&R rules* on the incoming event stream.

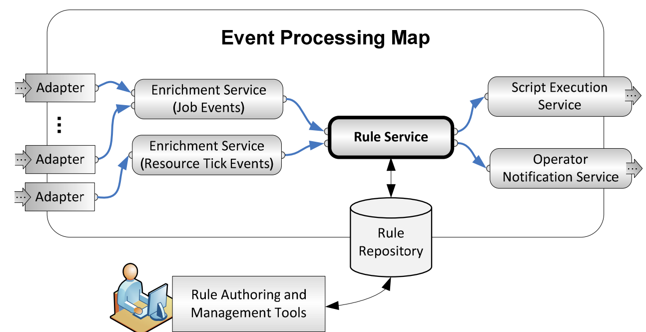
Response events are finally routed to one or more response adapters, which translate the response event into a format understandable to the underlying source system. A response event adapter could, for instance, add a message to a message queue, send an e-mail, or call an API.

Taken together, the following issues are addressed within a SARI application's event processing model:

- configuration of event services for processing steps
- interfaces to external systems for receiving data (sense) and for responding by executing business transactions (respond)
- event data transformations, event data analysis and persistence.

Figure 10 shows an exemplary event processing map for integrating and processing events from a workload automation system. Data is collected and received from source-system-specific adapters, which capture data from a job-scheduling system. Incoming events are enriched in order to prepare the event data for the rule processing. A typical example would be the attachment of additional job data; consider, for instance, scenarios where a source event only holds a job ID while the job's estimated runtime is required for the downstream decision-making logic. The rule service processes the enriched events according to S&R rules, which generate response events when noteworthy event situations are detected. The fired response events are published on the output port of the rule service and forwarded to response adapters. The response adapters transmit the response events to external systems by sending notifications or invoking a script in the workload automation system.

Figure 10 Event processing model with rule services (see online version for colours)



9 Example: intelligent workload automation

Modern IT landscapes have become highly complex environments, encompassing hybrid platforms and approaches, resources that are spread geographically and among a variety of different platforms, workloads that are allocated dynamically, and processes that change in response to new demands. As IT applications move to virtual or cloud environments, the need for intelligent automation software will become even greater.

More and more companies therefore begin to migrate from simple, time-based automation approaches ('start job *x* at *y* o'clock') to systems that support context awareness and the tracking of processes across the enterprise. Such systems automate tasks on an event-driven basis: 'start job *x* when *y* occurs'. Also, for maximum agility, companies need tools that can automatically and in real-time identify patterns in

the complex web of running processes, predict potential problems and needs before they become critical, and respond with corrective actions. Such tools are able to optimise the application and operational performance and maximise the efficiency of the resource utilisation.

In the following, we present a S&R rule for automatically controlling the jobs of a workload automation system. For our example, we assume that a rule shall monitor the resources of an IT environment (such as the CPU utilisation) and continuously check for delays and overload situations.

When modelling the rule for the above-stated business problem, we first have to identify the events which have to be evaluated for the rule processing, i.e., define the application's *event model*. For our example, we consider the following event types:

- *Job started*: Raised whenever a job is started on a host.
- *Job ended*: Raised whenever a job is stopped on a host.

'Job started' and 'job ended' events are correlated with a simple correlation set 'job', where events are linked if they refer to the same job via their 'job ID' attribute. The correlation set 'job' forms the basis for the example's pattern definition as described below.

Figure 11 shows the pattern definition for the above-described business scenario. It checks for CPU utilisation overheads and considerable overruns of estimated job-execution times. If these situations occur in parallel, an alarm signal 'delayed by overload' is raised. The signal provides access to the concerned job ID, the estimated and the real runtime of the job via output parameters. Also, the signal is configured with a silence interval of ten minutes. Input parameters allow configuring the concerned host ('host') and the critical delay in percent ('delay').

On the left-hand side of the decision graph, a business-object condition continuously checks the 'CPU utilisation' measure for the specified host. Measures of said measure type are updated centrally in another rule; an update of a measure implicitly triggers the evaluation of corresponding measure conditions within all pattern definitions of the SARI application.

On the right-hand side, two event conditions are used to identify delayed job executions on the specified host. In the prior condition, the overall set of jobs is limited to those running on the defined host. In the latter condition, the difference between the 'job ended' and the 'job started' creation time (i.e., the job's actual runtime) is compared to the estimated runtime of the job as available from the 'job started' event. If the job is delayed by more than the user-defined percentage, the condition evaluates to true. Please note that correlation aspects – ensuring that start and end events actually belong to each other – are not considered in the pattern model but follow from the underlying correlation model.

Table 1 and Table 2 show possible action definitions. For our example, we permit users to receive an e-mail notification or to reserve additional CPU resources on the target host by running a script on that host.

Figure 11 Pattern definition 'delay by overload' (see online version for colours)

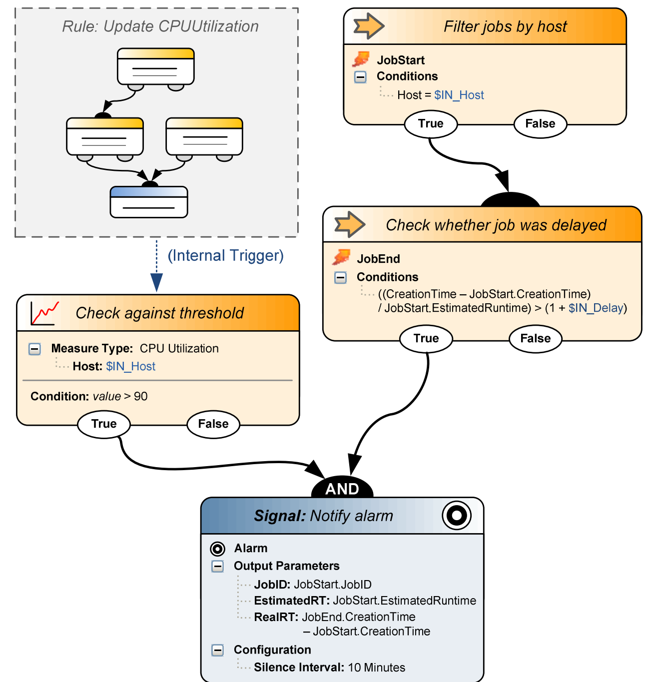


Table 1 Action definition 'notify administrator'

Input parameters	• Message (string) • Receiver (string)	
Description	E-mail alarm *Message* to *Receiver*	
Event type	E-mailEvent	
Event attribute	*Type*	*Expression*
Subject	String	'Alarm!'
Text	String	'Alarm Message: ' + $IN_Message
Receiver	String	$IN_Receiver
Priority	Integer	3

Table 2 Action definition 'add new resources to host'

Input parameters	• Number (integer) • HostName (string)	
Description	Reserve additional *Number* CPUs for host *HostName*	
Event type	ExecuteScriptEvent	
Event attribute	*Type*	*Expression*
Script	String	'…' + $IN_Number + '…'
HostName	String	$IN_HostName
User	String	'foo'
Password	String	'bar'

SARI features a web client for composing S&R rules from predefined pattern definitions and action definitions. In a first step, the user associates action definitions to the various signals of a pattern definition. In a second step, the user defines values for all input parameters. Finally, the user can activate the so-defined rule and add it to a rule service.

10 Experimental results

In the course of our research we conducted several experiments to evaluate the performance of S&R rules in real-world business scenarios. The findings highlight differences in processing rules of different types and quantities, and illustrate the performance implications of rule processing within a distributed execution environment. In the following, we compare the evaluation performance for sets of simple rules and sets of complex rules, where in both cases, the number of rules (rule sets of 10 rules vs. rule sets of 100 rules) and the number of execution nodes (single-node environments vs. two-node environments) is varied. For the presented measurements, systems with 4 dual-core CPUs and 8 GB of memory were used.

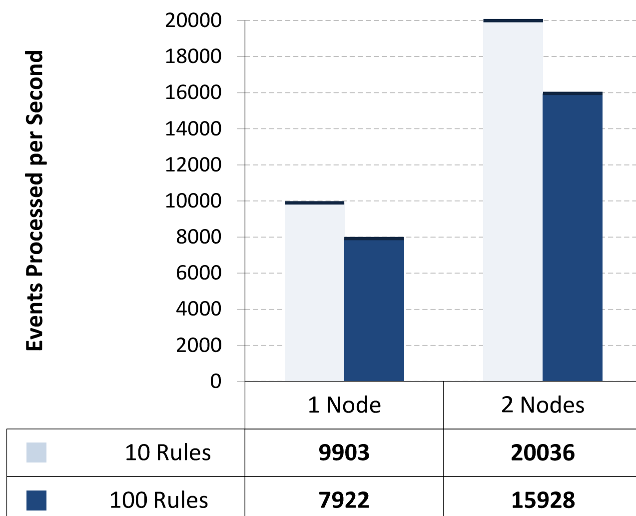
10.1 Processing simple rules, without event correlation

Our first experiment concerns the evaluation of simple rules that do not require event correlation; as described in Section 7, a so-defined decision graph is evaluated independently for each incoming event. In detail, the investigated rule sets are configured to satisfy the following conditions:

- total number of triggering event types: 10
- average number of attributes considered in a rule: 5
- no event correlation
- typical rule example: discovery of abnormally ended jobs or agents.

Figure 12 shows the results of our evaluation. Given a rule set of 10 rules and a single execution node, SARI's rule engine was able to process 9,903 events per second. The results show that the throughput depends on the number of rules being processed. Adding additional nodes to the system increases the throughput nearly linearly.

Figure 12 Processing simple rules without event correlation (see online version for colours)



10.2 Processing complex rules, with event correlation

Our second experiment focused on complex rules that require the usage of event correlation to yield meaningful results. In our evaluation, rules of the smaller rule set are based upon five different correlation sets in total, and rules of the larger rule set are based upon 8 different correlation sets in total. Besides, the investigated rules sets are configured to satisfy the following characteristics:

- total number of triggering event types: 10
- average number of attributes considered in a rule: 7
- event correlation: events correlated up to 8 dimensions (job, job type, job plan, agent, agent type, host, host group, client)
- 10% of all rules include aggregation functions over correlated events
- 10% of all rules include business-object updates and/or queries
- typical rule example: correlation of job events for hosts, calculation of an average error rate and discovery of high average error rate by host.

Figure 13 Processing complex rules with event correlation (see online version for colours)

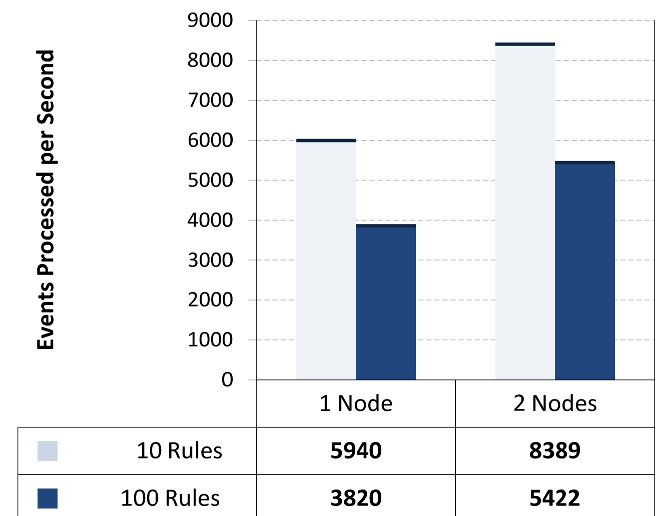


Figure 13 shows the results of our evaluation. On a single execution node, SARI's rule engine was able to process 5,940 events for the smaller rule set and 3,820 events for the larger one. The further decrease in throughput observed with a larger number of rules is caused by additional expenses for event correlation and the evaluation of aggregation functions on so-defined groups of events, as well as for updating and querying business objects. Adding an additional execution node to the SARI system increased the throughput to 8,389 events and 5,422 events, respectively. From our experiments, we determined that using event correlation on multiple execution nodes incurs a performance overhead of about 30% resulting from

node-to-node communications when sharing correlation sessions. This is necessary since events of a common correlation session may initially be processed on different execution nodes. For more details on the use of correlation sessions for synchronisation purposes, the interested reader may refer to McGregor and Schiefer (2004).

11 Conclusions and outlook

Today's networked business environments require systems which are adaptive and easy to integrate. Event-based systems have been developed and used to control business processes with loosely coupled systems. In this article, we presented a model-driven approach to building S&R rules for an event-based system.

By handling various aspects of an event-based application in separate, decoupled sub-models, SARI aims for expressiveness as well as manageability: The *event model* defines the structure of all possible event data. Relationships between incoming business incidents – e.g., whether two events belong to the same real-world business-process instance – are defined in the *correlation model*. The *business object model* defines virtual representations of real-world business objects such as customers, and allows for managing business state in a controlled and easy-to-handle manner. The *rule model* defines S&R rules, which associate noteworthy event situations – so-called event patterns – with appropriate reaction logic. An event situation may comprise sets of correlated events and business objects. Orchestration and integration aspects finally are handled in the *event processing model*.

S&R rules are the key element of a SARI application. In the event pattern model, pattern definitions are created from easy-to-understand pieces of pattern-detection logic – so-called rule components – which are combined with each other in a directed, acyclic decision graph. An event situation matches a so-defined pattern definition if it conforms to (at least) one path through the decision graph. Via a set of input parameters, the encapsulated pattern-detection logic can be adapted to different business scenarios. Signals and output parameters provide access to characteristics of a concrete event situation in a controlled and abstracted, e.g., to dynamically configure response actions. A graphical pattern-definition editor makes pattern modelling accessible to business users; exemplary rule-component shapes and pattern-definition renderings were presented throughout this article.

Taken together, the proposed approach simplifies the creation and management of event-triggered rules by means of:

- clean separation of concerns between an event, a correlation, a business-object, a pattern and an action model
- creation of complex pattern-detection logic from easy-to-understand pieces, via a graphical editor
- parameterisation and free composition of (encapsulated) event-processing logic, making it reusable across business scenarios and higher-level pattern definitions (i.e., hierarchical pattern modelling).

The work presented in this article is part of a larger, long-term research effort aiming at the development of a rule-management system for event-based systems. The key focus of this future research work is on modelling and managing comprehensive rule libraries for industry solutions. Also, we want to add forecasting components to SARI which shall allow business users to recognise emerging event patterns in order to proactively trigger counteractions.

References

Abadi, D.J., Carney, D., Çetintemel, U., Cherniack, M., Convey, C., Lee, S., Stonebraker, M., Tatbul, N. and Zdonik, S. (2003) 'Aurora: a new model and architecture for data stream management', *VLDB Journal*, Vol. 12, No. 2, pp.120–139.

Abadi, D.J., Ahmad, Y., Balazinska, M., Çetintemel, U., Cherniack, M., Hwang, J.H., Lindner, W., Maskey, A.S., Rasin, A., Ryvkina, E., Tatbul, N., Xing, Y. and Zdonik, S. (2005) 'The design of the Borealis stream processing engine', *Proceedings of the Conference on Innovative Data Systems Research*, pp.277–289.

von Ammon, R., Emmersberger, T., Greiner, T., Paschke, A., Springer, F. and Wolff, C. (2008) 'Event-driven business process management', *Proceedings of the Int. Conference on Distributed Event-Based Systems*.

Bailey, J., Crnogorac, L., Ramamohanarao, K. and Sondergaard, H. (1997) 'Abstract interpretation of active rules and its use in termination analysis', *Proceedings of the Int. Conference on Database Theory*, pp.188–202.

Baralis, E. and Widom J. (1994) 'An algebraic approach to rule analysis in expert database systems', *Proceedings of the Int. Conference on Very Large Data Bases*, pp.475–486.

Barbará, D., Mehrota, S. and Rusinkiewicz, M. (1994) 'INCAS: a computation model for dynamic workflows in autonomous distributed environments', Technical report, Department of Computer Science, University of Houston.

Bussler, C. and Jablonski S. (1994) 'Implementing agent coordination for workflow management systems using active database systems', *Proceedings of the Int. Workshop on Active Database Systems*, pp.53–59.

Chen, S-K., Jeng, J-J. and Chang, H. (2006) 'Complex event processing using simple rule-based event correlation engines for business performance management', *Proceedings of the Int. Conference on E-Commerce Technology and the Int. Conference on Enterprise Computing, E-Commerce, and E-Services*.

Dayal, U., Hsu, M. and Ladin, R. (1990) 'Organizing long-running activities with triggers and transactions', *SIGMOD Rec*,, Vol. 19, No. 2, pp.204–214.

Esper, Available at http://esper.sourceforge.net.

Geppert, A. and Tombros, D. (1998) 'Event-based distributed workflow execution with EVE', *Proceedings of the IFIP international Conference on Distributed Systems Platforms and Open Distributed Processing*, pp.427–442.

Haeckel, S. (1999) *Adaptive Enterprise: Creating and Leading Sense-and-Respond Organizations*, Harvard Business School Press.

Luckham, D. (2005) *The Power of Events*, Addison Wesley.

Ludascher, B. (1998) 'Integration of active and deductive database rules', PhD thesis, University of Freiburg, Germany.

McGregor, C. and Schiefer, J. (2004) 'Correlating events for monitoring business processes', *Proceedings of the Int. Conference on Enterprise Information Systems*, pp.198–205.

Rozsynai, S., Schiefer, J. and Schatten, A. (2007) 'Concepts and models for typing events for event-based systems', *Proceedings of the Int. Conference on Distributed Event-Based Systems*, pp.62–70.

Rozsnyai, S. (2008) 'Managing event streams for querying complex events', PhD thesis, Vienna University of Technology, Austria.

Schiefer, J., Obweger, H. and Suntinger, M. (2009) 'Correlating business events for event-triggered rules', *RuleML*, pp.67–81.

Schiefer, J. and Seufert, A. (2005) 'Management and controlling of time-sensitive business processes with sense & respond', *Proceedings of the Int. Conference on Computational Intelligence for Modelling, Control and Automation*, pp.77–82.

Suntinger, M., Obweger, H., Schiefer, J. and Gröller, M.E. (2008) 'The event tunnel: exploring event-driven business processes', *Computer Graphics and Applications*, Vol. 28, No. 6, pp.46–55.

Seiriö, M. and Berndtsson, M. (2005) 'Design and implementation of an ECA rule markup language', *RuleML*, pp.98–112.

Wu, P., Bhatnagar, R., Epshtein, L., Bhandaru, M. and Shi, Z. (1998) 'Alarm correlation engine (ACE)', *Proceedings of the Network Operations and Management Symposium*, pp.733–742.

World Wide Web Consortium (2007) SOAP 1.2, available at http://www.w3.org/TR/soap12.

Zdonik, S., Stonebraker, M., Cherniack, M. Cetintemel, U., Balazinska, M. and Balakrishnan, H. (2002) 'The Aurora and Medusa projects', *IEEE Data Engineering Bulletin*, Vol. 26, No. 1.

Notes

1 Unless otherwise stated, the described attribute-type model will apply to all (non-event-) data types as discussed in the remainder of this article.

2 For further details on SARI's tailored expression language, the interested reader may refer to Rozsynai (2008).