

# Complex Event Processing *Off the Shelf* – Rapid Development of Event-Driven Applications with Solution Templates

Hannes Obwegger, Josef Schiefer, Martin Suntinger, Florian Breier, and Robert Thullner

**Abstract**—Complex Event Processing (CEP) enables companies to monitor and control business processes in near real time. Despite the vast potential of generic CEP frameworks, experience showed that adopters are often overwhelmed with the inherent complexity of CEP, and heavily rely on external expertise to set up the required event-processing logic. In this paper we present a novel approach to creating, packaging and deploying event-based applications. Based on the event-processing framework SARI, so-called *solution templates* allow offering well-proven, standardized event-processing logic for common business needs. Delivering many benefits of off-the-shelf software while maintaining the flexibility of a CEP-based architecture, solution templates can easily be tailored to the specific business processes and needs of an individual corporation by customer-side domain experts. Real-time event processing logic is assembled from predefined, easy-to-use building blocks, in a way that fully abstracts from the underlying complexity. Event analysis is supported by predefined configurations for querying and visualizing historic event data. We demonstrate our approach using a real-world application from the workload automation domain.

## I. INTRODUCTION

COMPLEX Event Processing (CEP) [15] enables real-time monitoring of business incidents 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.

While there is little doubt about the vast potential of CEP, experience has shown that the industry’s original approach – namely, offering generic event-processing frameworks that can be used to develop arbitrary event-processing logic by customer-side technicians – is likely to overwhelm potential adopters of CEP. Early in the development of the discipline, many vendors therefore switched from a *framework-centered* to a *consulting-centered* business model, where vendors do not only offer a generic event-processing framework, but also deliver extensive expertise on how to actually use this framework. With vendor-side technicians developing tailored event-processing logic based on the individual requirements of an adopter, customers then pay for the resulting application rather than the generic CEP technology it is actually based on.

Despite manifold benefits for the early stages of CEP adoption, customer-level application development suffers from significant drawbacks, beginning with lengthy set-up phases and ending with difficult maintenance due to the high

number of solutions distributed among customers. As a consequence, CEP vendors strive to evolve their business models from consulting-centered to *solution-centered* approaches, where standardized, domain-specific event-processing applications are offered to ranges of customers with similar business needs. In this way providing CEP “off the shelf”, vendors as well as customers can benefit from the common advantages of COTS software, including rapid deployment, reliability, maintainability, regular and standardized updates, and application-level documentation and support. Despite these opportunities, state-of-the-art CEP frameworks fail to provide adequate mechanisms for packaging and deploying CEP logic in a way that enables customers to autonomously set up an application and align it with the company’s individual requirements and processes.

In this paper we present a novel approach to creating, packaging and deploying standardized event-processing applications based on the generic CEP framework SARI [19]. We propose solution templates, which are prepared based on the common requirements of a certain business domain and delivered “off the shelf” to customers. Besides encapsulating the basic event-processing logic of an application, solution templates provide diverse mechanisms for customer-side domain experts to set up, configure and analyze the delivered application in accordance with the specific needs of an individual corporation, in a way that fully abstracts from underlying complexity. The setup of a solution template is supported by wizard-based configuration dialogs. For the definition of customer-specific event-processing logic, rule spaces provide users with configurable, Lego-like building blocks of pattern detection and reaction logic. Domain experts assemble these blocks to concrete rules of the form “*if pattern then action(s)*”, which are automatically and transparently deployed as part of the application. For the ex-post analysis of historic event data, *visualization templates* provide preconfigured search and rendering configurations for data that are assumed to provide significant insights to both the business environment and the impact of user-defined decision making logic.

We present our approach using a real-world CEP application for service assurance in the workload automation domain, where we use SARI as an extension to the *UC4 Automation Platform* [22]. We understand service assurance as the proactive monitoring of business environments with the goal of detecting fault patterns and ensuring reliability and performance in a system landscape. It builds upon event data from both the technical infrastructure (including individual hosts, databases, and network connections) and the application layer of a corporation, which, besides the central automation platform, typically includes other enterprise systems such as ERP or CRM. Possible higher-level business logic is not currently considered.

Manuscript received February 15, 2011.

H. Obwegger, J. Schiefer, M. Suntinger and F. Breier are with UC4 Se-nactive Software GmbH, Vienna, Austria (e-mail: {hannes.obwegger, josef.schiefer, martin.suntinger, florian.breier}@uc4.com)

R. Thullner is with Secure Business Austria, Vienna, Austria (e-mail: rthullner@sba.at)

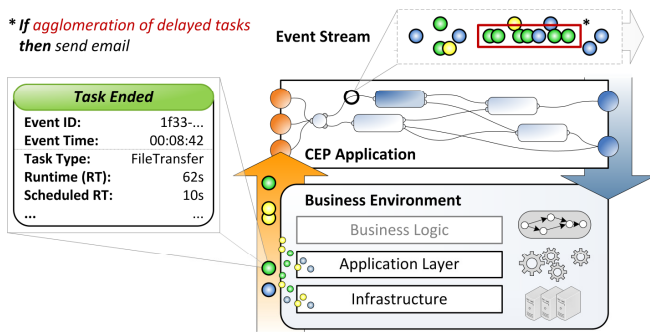


Figure 1. Event-based service assurance

Figure 1 sketches an event-based approach to service assurance. A CEP application retrieves events for all noteworthy incidents in the business environment. In various parts of the application, event-pattern rules are applied on the incoming event stream to detect relevant patterns, e.g., an uptrend in application errors or execution delays. In response to such patterns, the CEP engine proactively intervenes in the business environment, e.g., by temporarily allocating additional resources, throttling uncritical business tasks or notifying system administrators.

The remainder of this paper is structured as follows: In Section II, we discuss related work. In Section III, we present an architectural overview to the concept of solution templates and their integration with a customer’s IT landscape. An exemplary solution template for service assurance is discussed in Section IV. The solution template’s rule spaces and visualization templates are discussed in Section V and Section VI. Section VII summarizes this paper and gives an outlook to future work.

## II. RELATED WORK

Since introduced to a wider community by David Luckham and his seminal work on “The Power of Events” [15], Complex Event Processing has inspired numerous projects of both academic (e.g. [1][2][20]) and commercial nature (cf. [8]) as well as in the Open Source community [5].

While there is some work on tools that help application developers to design and validate event-pattern rules (e.g. [5]), there is little work on the broader issue of event-based application development, the packaging of such applications and their deployment at customers. Magid et al. [12] propose an approach where based on a set of rules for the generic event-processing framework AMiT, an application in the form of domain-specific, optimized byte code is generated. While doubtlessly useful in domains where the focus lies on performance rather than adaptability, we believe that the flexibility of an event-based architecture brings great advantages when the underlying business environment is likely to change, or later adaptations may be desired. While generally designed to be deployed “off the shelf”, solution templates remain open for adaptations by vendor-side experts or customer-side power users.

Solution templates allow domain experts to model high-level event-processing logic from prepared, Lego-like building blocks without having to understand (or even know) the event-based foundation of decision making. In the community, making event processing accessible to business users is seen as a key issue in the future development of CEP: Etzion and Niblett [7] see the development from programming-centred to semi-technical development tools as one of the emerging directions in event processing. Chandy and Schulte [4] identify the ability to “enable business users to tailor systems to their needs” as a major criterion for the relevance of event-based systems, arguing that a one-size-fits-all specification of events and responses doesn’t work. Mismatches between the complexity of CEP and the abilities of its adopters have already inspired the definition of domain-specific reference models [1] and an approach to automated rule parameter prediction and correction [16].

On the event-analysis side, solution templates support the mining of noteworthy patterns from historic event data through so-called *visualization templates*. Long before CEP stepped into the spotlight, business users have been recognized as a primary user group for data mining tools. “Verticalization” means the incorporation of domain-specific knowledge into analytical frameworks (cf. [12]); providing search and visualization configurations for data that are assumed to be useful with respect to a given domain, visualization templates in this way simplify the usage of SARI’s generic event-analysis framework.

The application of CEP in industrial settings has been discussed in a number of experience reports, ranging from finance [9] [14], to RFID tracking [14], and automation [11]. In this paper we present CEP in the context of *service assurance*, which we understand as the proactive monitoring of business environments with the goal of detecting fault patterns and ensuring the reliable, high-performing operation of a system landscape. As such, service assurance includes – yet, is not restricted to – disciplines such as Business Service Management (BSM) and Application Performance Management (APM). Service assurance in the described sense has, for instance, been discussed in the context of cloud computing [17].

## III. ARCHITECTURAL OVERVIEW

Figure 2 shows an overview to the proposed architecture for “CEP off the shelf”. It is separated horizontally, into *event processing* and *event analysis*, and vertically, into layers ranging from the customer’s IT landscape on bottom to the actual users of an application – so-called *business operators* – on top.

The bottom layer of the architecture shows the customer-side IT landscape, which primarily consists of the customer’s IT infrastructure, diverse enterprise software and the central automation platform. In addition, the customer must provide a database for storing analytical event data, which we refer to as the Event Data Warehouse (EDWH) in the remainder of this paper.

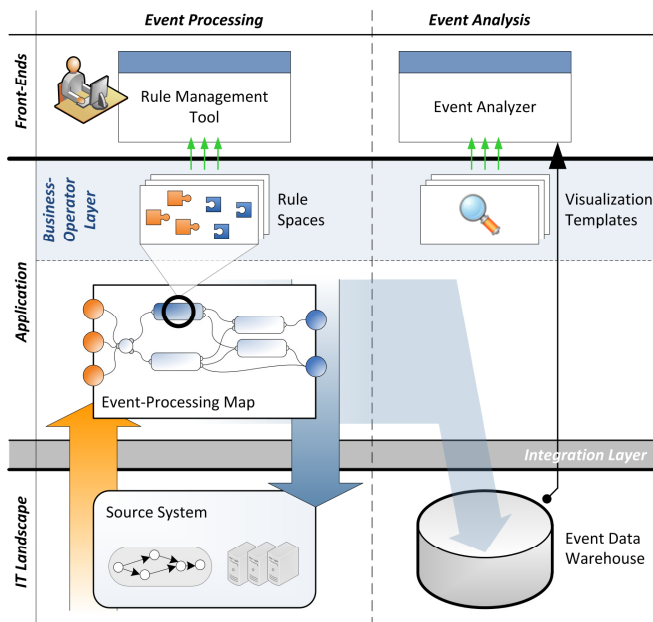


Figure 2. SARI application architecture.

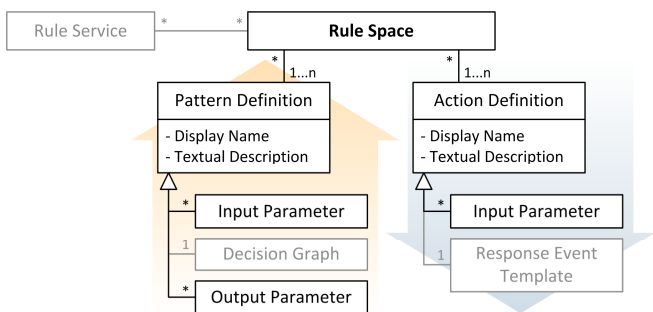


Figure 3. Rule space meta-model.

As an intermediate layer between the IT landscape and the actual event-processing application, a conceptual *integration layer* represents the customer-specific integration of event-processing logic with the underlying business environment; for instance, pulling event data from a database requires a customer-specific connection string, and sending an e-mail requires access to the customer’s in-house SMTP server. Solution templates come with wizard-based configuration dialogs, which guide customer-side technicians through the initial setup of a solution template, using a domain-specific terminology where possible.

The actual *solution template* can be further separated into the *event-processing infrastructure*, which is typically hidden from customer-side end users, and a *business-operator layer*, which serves as an interface to the business operator and is directly accessible through the provided collection of front-end tools.

The central concept of any template’s event-processing infrastructure is the *event-processing map*, a predefined orchestration of *event adapters* and *event services*. Event adapters may be considered the actual interface to the underlying source system: Depending on their implementation,

event adapters translate real-world actions (such as a user *actually* placing a bet in an online gambling platform) into event representations of a certain event type, and vice versa. Event services receive events from event adapters or other event services, process them based on implementation-specific logic, and respond back to the map.

The so-defined event-processing infrastructure of an application template contains all basic event processing that is required independently from the concrete, customer-specific monitoring needs. Besides providing the basic facilities for retrieving events and responding back to the source system, this typically includes data enrichment, filtering of unneeded or erroneous data, as well as basic event data aggregations. Also, the event-based infrastructure typically defines the persistence of selected event data for analysis purposes.

While in general hidden from customer-side end users, the event-processing infrastructure of a solution template exposes a collection of *rule spaces*, which constitute the event-processing part of the business-operator layer. Rule spaces organize Lego-like building blocks of a.), pattern-detection logic (so-called *pattern definitions*), and b.), reaction logic (so-called *action definitions*) based on the specific organizational tasks to which they belong. Through a tailored, web-based rule management client [10], business operators may then use the building blocks of a rule space to assemble concrete event-pattern rules of the form “if *pattern* then *action(s)*” in a way that fully abstracts from the event-based foundations of decision making. Having defined such a rule, it is automatically deployed on one or more associated *rule services* in the underlying event-processing map. Rule services are special event services that apply collections of rules on incoming event streams and publish response events via their output ports. In the following, we refer to rule services that host rule spaces as *rule space executors*.

Figure 3 shows the meta-model for rule spaces. Pattern definitions are based on decision graphs, which describe noteworthy classes of event situations in a graph-based pattern language [17]. Action definitions specify response-event templates, which define the structure of a response event to be generated when the action is triggered as part of a rule. Both kinds of building blocks are configurable through a collection of typed, named and documented *input parameters*. Being used as typed placeholders for concrete values in the encapsulated piece of event-processing logic, input parameters must be specified before a building block can be used as part of a rule. Pattern definitions furthermore expose a collection of *output parameters*, which provide access to selected characteristics of a triggering event situation in a controlled and fully abstracted manner. Both kinds of building blocks eventually provide a high-level, natural-language description of the encapsulated event-processing logic, with placeholders for all input parameters. By definition, these descriptions begin with the term “if” and “then”, respectively. In the rule management client, business operators then assemble rules as natural-language sentences from so-defined clauses.

On the event analysis side of the architecture, business operators may investigate historical event data using the *Event Analyzer* [21], a rich visualization and data mining framework tailored to the specific characteristics of (possibly complex) events. To shorten the navigation through potentially vast amounts of event data, business operators are provided with so-called *visualization templates* for event data that are assumed to be relevant in the given domain. Visualization templates encapsulate a query on the underlying EDWH along with diverse visualization configurations, including the coloring and placement of event glyphs. As with rule building blocks, visualization templates expose a collection of typed *input parameters*, which must be specified by the business operator when applying a template.

#### IV. APPLICATION OVERVIEW

In the following sections, we demonstrate the proposed approach to “CEP off the shelf” using a real-world application template for service assurance in the workload automation domain. Requiring only minor customer-specific adjustments (if any), it has been successfully applied in a variety of business environments, ranging from SMBs to major corporations.

Table I list the various *event types* of the proposed solution template, roughly separated into input events (coming from the source system), virtual events (being created during event processing), and response events (being send to response event-adapters and transformed to real-world response actions). The solution template’s *event-processing map* is shown in Figure 4; for the sake of readability, input ports, output ports and connections are colored according to the event type they are concerned with. On the left-hand side of the figure, a collection of sense event-adapters receive task events, message events and log-file events from the underlying source system. All events are forwarded to a “Message Enrichment” service, where based on event-attribute values, additional data are retrieved from external sources and attached to events. Enriched task events are then forwarded to the “System Checkpoint Generator”, where system checkpoint events are generated for each x incoming task events. Enriched log-file events are forwarded to the “Database Error Detector” service, which detects from all log-file events those that signify database-related entries and transforms them to database log-file events. (Forming a branch from the main event flow, task and message events are also forwarded to a “Statistics” service, where application-wide aggregations are calculated.) Before being collectively forwarded to a grouping of rule space executors, all events are written to the EDWH for ex-post analysis. (Due to their enormous amount compared to message and log-file events, tasks events undergo a filtering before being persisted; from all incoming task events we actually store only those with exceptional runtimes or noteworthy end states.) The various rule space executors host a collection of rule spaces as will be discussed in Section V. Although not depicted in Figure 4, each rule service is provided with exactly

TABLE I  
EVENT TYPES FOR SERVICE ASSURANCE

<b>Input Events</b>	<p><b>Tasks Events</b> represent the end of a task in the underlying automation platform, and form the major input for detecting situations such as delayed task executions, manual cancellations of tasks, or recurring task failures.</p> <p><b>Message Events</b> represent noteworthy occurrences as are signified by the automation platform; for instance, a message could indicate the shutdown of a task-execution agent.</p> <p><b>Log-File Events</b> are semantically related to message events, however, represent data as is written to the application’s log files and contain detailed system-trace information such as execution times of system-internal activities or database transactions.</p>
<b>Virtual Events</b>	<p><b>System Checkpoint Events</b> are generated within the application, and basically form an aggregation of a configurable number of incoming task events. These events contain, for instance, the number of failed tasks among the last x tasks, the average overtime, etc. Reducing the overall number of task data, system checkpoint events can well be used for detecting trends.</p> <p><b>Database Log-File Events</b> are special log-file events that signify database-related entries. In addition to the basic log-file event data, they provide fields for the database error number, etc.</p>
<b>Response Events</b>	<p>A collection of response events are provided for responding back to the underlying source system. These response events include <b>Email Events</b>, <b>Log-File Entry Events</b> (which cause a respective event adapter to write an entry to log file), <b>Command-Line Action Events</b>, and <b>Web-Service Call Events</b>.</p>

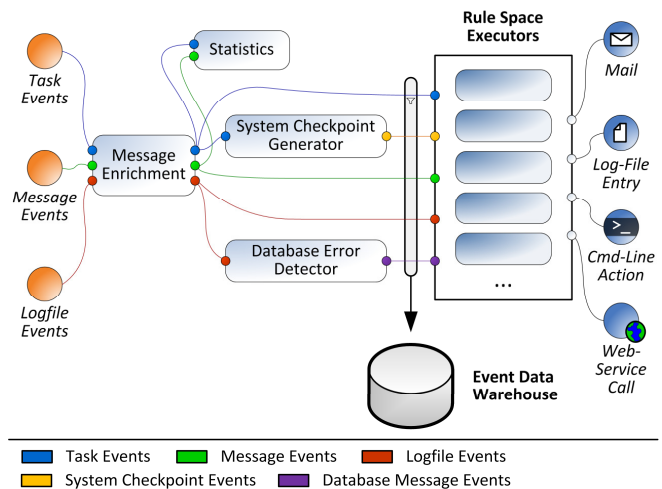


Figure 4. Event-processing map for application assurance.

those events that are potentially relevant for the provided pattern-detection logic. On the output side of the application, a collection of response event-adapters allows sending emails, writing to log files, executing arbitrary command-line actions, and web-service calls. Again, rule services are connected to all event adapters that are required for the provided set of action definitions.

#### V. TEMPLATE-BASED EVENT PROCESSING

In Section IV we have discussed the event-processing infrastructure of a real-world solution template for service assurance. In the following, we present the event-processing side of the template’s *business-operator layer*, which is formed by a collection of *rule spaces* for the various operational tasks within an application. While giving an introduction to the complete list of rule spaces and their specific

roles in an application, we will particularly focus on a selected rule space “runtime monitoring” – basically concerned with the detection of tasks that outrun their scheduled execution time – when it comes to concrete examples.

### A. Rule Spaces Overview

Solution templates expose rule spaces for the diverse operational tasks within an application. The presented solution template for service assurance offers six rule spaces to business operators, where each is concerned with a different aspect of the underlying automation platform: *Runtime monitoring*, *access denial monitoring*, *log file and error message monitoring*, *agent monitoring*, *health monitoring*, and *database protection monitoring*. Table II provides an introduction to said rule spaces and their specific role in an application.

### B. Pattern Definition Example

In the presented solution template, the rule space for *runtime monitoring* provides diverse pattern definitions based on the occurrence of delayed tasks, where the actual execution time outruns the scheduled execution time by more than a specified percentage. Specialized pattern-detection logic allows detecting delays on certain hosts, at certain days of a week, or at certain times. With large companies typically focusing on recurring delays rather than individual outliers, other pattern definitions allow specifying a certain number of runtime exceedances to be reached; depending on the pattern definition, the concerned tasks must occur in total or within a specified time frame.

Table III shows the exemplary “Recurring delays per host” pattern definition. The pattern definition exposes two input parameters “Runtime delay” (to be specified in percent) and “Number of exceedances”, as well as an output parameter “Host” providing access to the specific host on which the pattern was detected. The encapsulated decision graph checks in an initial condition on “task” events whether a task exceeds its scheduled runtime by more than the specified percentage. If this is the case, a pattern-internal score “Task runtime exceedances” is incremented for the concerned host. If the resulting score value is greater than the specified number of exceedances, a special signal component is activated; signals indicate the detection of an event situation and calculate concrete values for a pattern definition’s output parameters. In addition, the score is reset.

### C. Action Definition Example

On its action-definition side, the runtime-monitoring rule space provides diverse email-based actions for notifying responsible departments within a company. As overload situations may often be resolved automatically through additional resources or rescheduling of tasks, the rule space furthermore provides a variety of so-called *system actions*: Based on a web-service call, they directly feed back into the automation platform, where they may request machines, pause or cancel running tasks, or delay the execution of scheduled ones.

TABLE II  
RULE SPACES FOR SERVICE ASSURANCE

<b>Runtime monitoring.</b> An important indicator for application assurance is the task runtimes. The runtime-monitoring rule space allows monitoring task runtimes for exceptional delays. Details on the rule space’s pattern and action definitions will be discussed in greater detail in Section V.B, Section V.C, and Section V.D of this paper.
<b>Access denial monitoring.</b> Messages about access denials on objects of the automation engine are important hints for security violations. This rule space allows detecting conspicuities in access denial messages, such as a certain user trying to access an object unsuccessfully multiple times within a short period, or with recurring retries after a break.
<b>Log file and error message monitoring.</b> In case of a failure, manifold information is available in the server log files indicating what led to the problem. This rule space allows analyzing these log files in real-time, to detect early indicators for system troubles. Frequent errors are recognized as well as an increase in the total number of error log entries. Known messages and indicators might also trigger an immediate alert.
<b>Agent monitoring.</b> Agents need to be constantly up and running in order to execute the scheduled processes. This rule space allows monitoring agents for availability and performance. Shutdown/failure patterns are identified as well as trouble agents with frequent shutdowns or insufficient response times. In addition, it allows monitoring if certain tasks fail in recurring fashion on an individual host.
<b>Health monitoring.</b> Indicators for system health include execution delays (indicating unbalanced load), high rates of task failures or cumulations of error messages, among others. While individual triggers might not yet alert, trends in the number of failures or delays might show changes in the application stability. Health monitoring rules target exactly these trends, alerting upon increasing numbers of also minor delays, slight increases in error occurrences and task failures or a combination of all three.
<b>Database protection monitoring.</b> The automation engine keeps track on every action in an underlying database. Also, messaging is relayed over the database to keep it reliable and persistent. Thus, the database performance is vital for the performance of the automation engine. This rule space allows monitoring in real-time for indicators on potential database issues. These include log entries on critical, long-running database calls, error messages, timeouts, or even deadlocks.

TABLE III  
EXEMPLARY PATTERN DEFINITION

Input Parameters	ID	Type	Variable Name
	Runtime delay	Integer	\$delay
	Number of exceedances	Integer	\$exceedances
Description	(if <i>Number of exceedances</i> delays of more than <i>Runtime delay</i> percent occur on a host		
Decision Graph			
Output Parameters	ID	Type	Expression
	Host	String	Task.Host

TABLE IV  
EXEMPLARY ACTION DEFINITION

Input Parameters	ID	Type	Variable Name
	Object	String	\$Object
Description	(then) start <i>Object</i>		
Response-Event Template	<b>Response-Event Type:</b> com.example.common.WebServiceAction		
	Event Attribute		
	ID	Type	Expression
	WSDL	String	\$\$WSDLPath
	Method	String	“executeObject”
	Args	List	{ \$Object, “UTC”, Now(), ... }
	...	...	...

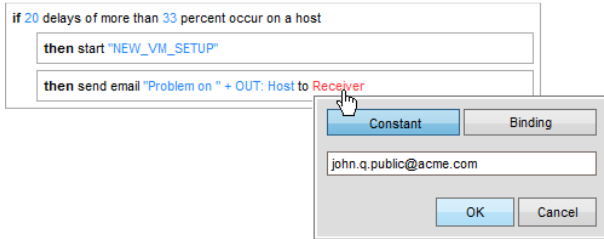


Figure 5. Rule creation using the rule-management web client.

Table IV shows an exemplary action definition “Start additional host”. Exposing a string-typed input parameter “Object”, it eventually results in a respectively-configured “Web Service Action” event, where the “Arguments” attribute is calculated dynamically from the provided object name. While the method name is provided as a constant, the exact path to the web service’s WSDL is read from an application-wide resource string. Setting such resource strings is part of the customer-specific integration of a solution template with the underlying IT landscape and hence supported by tailored integration wizards as provided as part of the solution template’s *integration layer*.

#### D. Rule Example

To define specific monitoring logic based on the requirements of the current business situation, business operators eventually assemble the building blocks of a rule space to concrete rules of the form “*if pattern then action(s)*”. In SARI, business operators are therefore provided a web-based user interface that fully abstracts from the event-based foundations of decision making. Figure 5 shows a screenshot of the web client: Having selected the rule space and the pattern definition in previous steps of a rule creation wizard, the business operator now associates the pattern definition with one or more action definitions and specifies concrete values for all input parameters. In the shown example, the business operators configures the rule’s pattern to consider tasks with a delay greater than 33% and trigger whenever 20 such tasks are detected on a host. In this case, a task “NEW\_VM\_SETUP” shall be started in the automation platform. In addition, an email (including the concerned host) shall be sent to a system administrator.

TABLE V  
VISUALIZATION TEMPLATES FOR SERVICE ASSURANCE

<b>Message Analysis.</b> Message events contain manifold information such as agents’ start and stop times, error states or time critical database calls. Message-analysis visualization templates give an overview to such events and may serve as a starting point for drilling down into detail.
<b>Agent Analysis.</b> Especially in large systems gaining an overview on agents’ performance is crucial. Tailored visualization templates provide insight into message on agents which have been stopped, started or ended abnormally in order to recognize regular faults and error patterns.
<b>Manual Interaction Analysis.</b> Manual interactions, such as manual cancellations of tasks, might have big influence on the whole system and be the reason for future failures. Tailored visualization templates provide insights to so-defined interactions and possible consequences thereof.
<b>Access Denial Analysis.</b> With multiple users operating in a system, security becomes an issue. In the proposed solution, any attempt to gain unauthorized access to a system is logged and can be investigated <i>ex posteriori</i> with respective visualization templates.
<b>System Overview.</b> Over the course of several days or weeks, potentially millions of tasks might be executed. In such case, gaining a big picture of the system on the level of single tasks is impossible. System overview visualization templates provide insights based on system checkpoints events, which summarize execution statistics for each $x$ finished tasks.
<b>Task Analysis.</b> When tasks exceed their estimated runtime or fail, it is essential to dig into details and mine possible patterns within these occurrences. Tasks analysis visualization templates provide queries for delayed and failed tasks and plot them from different perspectives.

## VI. TEMPLATE-BASED EVENT ANALYSIS

In the presented architecture, rule spaces allow tweaking the real-time event-processing logic of an application to meet the specific requirements of a corporation. Serving as a major input for the definition and continuous improvement of such logic, the *event analysis* part of SARI is concerned with the mining of noteworthy patterns in historical event data. In the following, we present the event-analysis side of the solution template’s business-operator layer. We give an introduction to the overall set of visualization templates and focus on a selected template “abnormally ended tasks per time of week” for a concrete example.

### A. Visualization Templates Overview

The presented solution template for service assurance provides an overall number of 40 visualization templates. According to the specific aspects of an application with which they are concerned, visualization templates are organized in a simple folder structure, into *message analysis*, *agent analysis*, *manual interaction analysis*, *access denial analysis*, *system overview*, and *object statistics*. Table V provides an introduction to said groups of templates and their specific role in an application. (Note that in contrast to rule spaces which associate event-processing logic with specific rule services to be executed on, the described grouping has no relevance for the actual functioning of a visualization template.)

### B. Visualization Template Example

From the various visualization templates concerned with *object statistics*, the “abnormally ended tasks per time of week” template provides insights to the temporal distribution

TABLE VI  
EXEMPLARY VISUALIZATION TEMPLATE

Input Parameters	ID	Type	Variable Name
	Start time	Time stamp	\$StartTime
	End time	Time stamp	\$EndTime
	User ID	Integer	\$UserID
Query	all task events in [\$StartTime, \$EndTime] with User ID = \$UserID and exceptional end status		
Coloring (Event Chart)	by end status		
Placement (Event Chart)	by time of day on x-axis, by day of week on y-axis		
...	...		

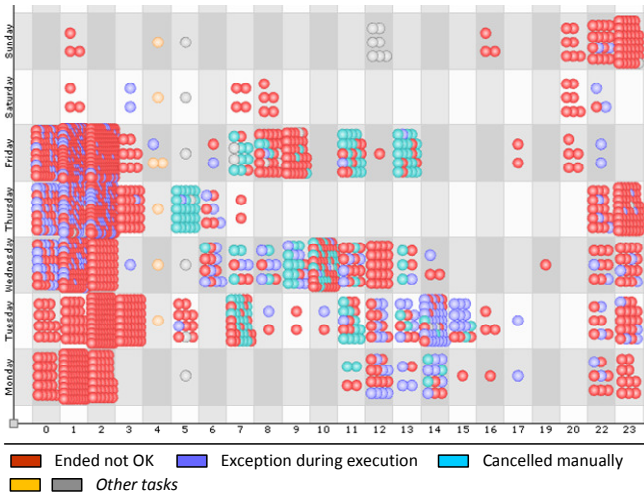


Figure 6. Template-based event-data visualization

of exceptional tasks – i.e., tasks that end with a status other than “okay”, are cancelled manually or raise any kind of exception during their execution – within a week. Table VI sketches the visualization template. It exposes two time-stamp-typed input parameters “Start Time” and “End Time”, specifying the time frame for which data shall be loaded, as well as an Integer-typed input parameter “User ID”, specifying the concerned user account. The template’s query loads from the EDWH all task events that are marked as abnormally ended and fulfil the described conditions. In the central Event Chart visualization, events are colored by their status and positioned based on the event occurrence’s day of week in vertical and the time of day in horizontal direction.

Figure 6 shows a rendering of the visualization template when executed for a time frame of two weeks and a selected client ID. It shows an accumulation of failed tasks during the night hours of working days, where from Wednesday to Friday, a high number of exceptions are raised during task executions. On Monday to Tuesday, exceptions occur primarily from noon to 2 pm. Tasks are cancelled manually regularly during working hours, with noticeable accumulations from Tuesday to Friday.

## VII. CONCLUSION

In this paper we presented a novel approach to creating, packaging and deploying domain-specific event-processing applications based on the generic event-processing framework SARI. It enables CEP vendors to deliver standardized event-processing logic “off the shelf” to customers in a certain business domain. Besides encapsulating the basic event-processing logic of an application, so-called *solution templates* provide diverse mechanisms for customer-side domain experts to set up, configure and analyze the application in accordance with the specific business needs of their application, in a way that fully abstracts from the underlying complexity. We demonstrated our approach using a real-world application for service assurance in the workload automation domain, which has been successfully applied in a variety of business environments.

The presented work is a part of a long-term research effort towards making Complex Event Processing accessible to business users with little or no technical expertise. It particularly focuses on the aspect of rule management, which showed to be too complex for domain experts in existing approaches. Future work will put the focus on the ex-post analysis of historical event data, where we are currently working on new visualization methods on different layers of abstraction, their presentation to business users and their integration with existing methods.

## REFERENCES

- [1] D. J. Abadi, D. Carney, U. Cetintemel, M. Cherniack, C. Convey, S. Lee, M. Stonebraker, N. Tatbul, and S. Zdonik, “Aurora: A new model and architecture for data stream management”, in *VLDB Journal*, vol. 12, pp. 120–139, 2003.
- [2] A. Adi and O. Etzion, “AMIT – The situation manager”, in *VLDB Journal*, vol. 13(2), pp. 177–203, 2004.
- [3] R. v. Ammon, C. Silberbauer, and C. Wolff, “Domain specific reference models for event patterns – for faster developing of business activity monitoring applications”, in *VIP Symposia on Internet related research with elements of M+I+T++*, 2007.
- [4] K. M. Chandy and W. R. Schulte, *Event Processing: Designing IT Systems for Agile Companies*. McGraw-Hill, New York, NY, 2010.
- [5] A. Ericsson and M. Berndtsson, “REX, the rule and event eXplorer”, in *Proceedings of the 2007 Int. Conference on Distributed Event-Based Systems*, pp. 71–74. ACM, New York, NY, 2007.
- [6] Esper, <http://esper.sourceforge.net>
- [7] O. Etzion and P. Niblett, *Event Processing in Action*. Manning Publications, Stamford, CT.
- [8] M. Gualtieri and J. R. Rymer, *The Forrester Wave™: Complex Event Processing (CEP) Platforms, Q3 2009*. Forrester Research, Cambridge, MA, 2009.
- [9] T. Greiner, W. Düster, F. Pouatcha, R. v. Ammon, H.M. Brandl, and D. Guschakowski, “Business activity monitoring of norisbank taking the example of the application easyCredit and the future adoption of Complex Event Processing (CEP)”, in *Proceedings of the 4th Int. Symposium on Principles and Practice of Programming in Java (PPPJ '06)*, pp. 237–242. ACM, New York, NY, 2006.
- [10] A. Kavelar, J. Obweiger, J. Schiefer, and M. Suntinger, “Web-based decision making for Complex Event Processing systems”, in *Proceedings of the 2010 6th World Congress on Services (SERVICES '10)*, IEEE Computer Society, Washington, DC, 2010.
- [11] I. Kellner and L. Fiege, “Viewpoints in complex event processing: industrial experience report”, in *Proceedings of the Third ACM Int. Conference on Distributed Event-Based Systems*. ACM, New York, NY, USA.

- [12] R. Kohavi, N. J. Rothleder, and E. Simoudis, "Emerging trends in business analytics", in *Commun. ACM*, vol. 45(8), pp. 45–48. ACM, New York, NY, 2002.
- [13] Y. Magid, A. Adi, M. Barnea, D. Botzer, and E. Rabinovich, "Application generation framework for real-time Complex Event Processing", in *Proceedings of the 2008 32<sup>nd</sup> Annual IEEE Int. Computer Software and Applications Conference (COMPSAC '08)*, pp. 1162–1167. IEEE Computer Society, Washington, DC, USA, 2008.
- [14] Y. Magid, G. Sharon, S. Arcushin, I. Ben-Harrush, and E. Rabinovich, "Industry experience with the IBM Active Middleware Technology (AMiT) Complex Event Processing engine", in *Proceedings of the Fourth ACM Int. Conference on Distributed Event-Based Systems*, pp. 140–149. ACM, New York, NY, 2010.
- [15] D. Luckham, *The Power of Events*. Addison-Wesley, 2002.
- [16] Y. Turchin, G. Avidgor, and S. Wasserkrug, "Tuning Complex Event Processing rules using the prediction-correction paradigm", in *Proceedings of the Third ACM Int. Conference on Distributed Event-Based Systems (DEBS '09)*. ACM, New York, NY, 2009.
- [17] R. Sandhu, R. Boppana, R. Krishnan, J. Reich, T. Wolff, and J. Zachry, "Towards a discipline of mission-aware cloud computing", in *Proceedings of the 2010 ACM Workshop on Cloud Computing Security (CCSW '10)*, pp. 13–18. ACM, New York, NY, 2010.
- [18] J. Schiefer, S. Rozsnyai, C. Rauscher, and G. Saurer, "Event-driven rules for sensing and responding to business situations". In *Proceedings of the Int. Conference on Distributed Event-Based System*, pp. 198–205, 2007.
- [19] J. Schiefer and A. Seufert, "Management and controlling of time-sensitive business processes with Sense & Respond", in *Proceedings of the Int. Conference on Computational Intelligence for Modelling Control and Automation*, pp. 77–82, 2005.
- [20] M. Seiriö and M. Berndtsson, "Design and implementation of an ECA rule markup language", in *RuleML*, pp. 98–112. Springer, Berlin, 2005.
- [21] M. Suntinger, H. Obwegger, J. Schiefer, and M. E. Gröller, "Event Tunnel: Exploring Event-Driven Business Processes", in *IEEE Comput. Graph. Appl.*, vol. 28(5), pp. 46–55, 2008.
- [22] UC4 Software Inc, UC4 Automation Platform, 2011.