MEdit4CEP: a model-driven solution for real-time decision making in SOA 2.0

Juan Boubeta-Puig*, Guadalupe Ortiz, Inmaculada Medina-Bulo

*Department of Computer Science and Engineering, University of Cádiz,
Avda. de la Universidad de Cádiz n° 10, 11519 Puerto Real, Cádiz, Spain*

Abstract

Organizations all around the world need to manage huge amounts of data from heterogeneous sources every day in order to conduct decision making processes. This requires them to infer what the value of such data is for the business in question through data analysis as well as acting promptly for critical or relevant situations. Complex Event Processing (CEP) is a technology that helps tackle this issue by detecting event patterns in real time. However, this technology forces domain experts to define these patterns indicating such situations and the appropriate actions to be executed in their information systems, generally based on Service-Oriented Architectures (SOAs). In particular, these users face the incommodity of implementing these patterns manually or by using editors which are not user-friendly enough. To deal with this problem, a model-driven solution for real-time decision making in event-driven SOAs is proposed and conducted in this paper. This approach allows the integration of CEP with this architecture type as well as defining CEP domain and event pattern through a graphical and intuitive editor, which also permits automatic code generation. Moreover, the solution is evaluated and its benefits are discussed. As a result, we can assert this is a novel solution for bringing CEP technology closer to any user, positively impacting on business decision making processes.

Keywords: decision making, complex event processing, model-driven development, graphical modeling editor, SOA 2.0

* Corresponding author. Tel.: +34 956 483307

Email addresses: juan.boubeta@uca.es (Juan Boubeta-Puig), guadalupe.ortiz@uca.es (Guadalupe Ortiz), inmaculada.medina@uca.es (Inmaculada Medina-Bulo)

Publisher version:

Boubeta-Puig, J., Ortiz, G., & Medina-Bulo, I. (2015). MEdit4CEP: A model-driven solution for real-time decision making in SOA 2.0. *Knowledge-Based Systems*, 89, 97–112. <https://doi.org/10.1016/j.knosys.2015.06.021>

1. Introduction

Nowadays, organizations all around the world need to manage huge amounts of data from heterogeneous sources every day in order to conduct decision making processes. Decision making is the process by which one action is chosen among others based on the additional benefits it can provide us with [1]. To be successful, a decision making process requires, among other factors, prompt information regarding what the value of such data is for the business in question. A thorough analysis of data, as well as fast reaction for critical or relevant situations, will allow the organization to be positioned above its competitors [2]. Nevertheless, it is a complex process since, among other reasons, data are heterogeneous –they do not share a common format– and they should be processed in real time. Therefore, a computerized decision support system may help human beings make real-world decisions [3], [4].

In this context, Complex Event Processing (CEP) [5] is a technology that allows the analysis and correlation of large volumes of data with the aim of detecting complex and meaningful events and of inferring valuable knowledge for end users. This knowledge will be really helpful in the decision making process. In order to do this, so-called event patterns are used. These patterns specify which conditions must be met in order to detect such situations of interest.

Despite the great advantages that CEP can bring to a business, it is a substantial challenge for users who are business experts, but do not have the necessary experience and they lack knowledge of how this technology is used. One of the main problems these users have to face is precisely the definition of these event patterns using particular languages, those called Event Processing Languages (EPLs). Although some current software solutions provide graphical tools in order to solve this problem [6]–[8], none of them are user-friendly enough, since they require users to hand-write at least a portion of the code for pattern definition.

On the other hand, current information systems tend to be based on Service-Oriented Architectures (SOAs) due to the fact that this type of software architecture allows the development of highly scalable distributed systems as well as their integration with own and third-party systems. These are being combined with Event-Driven Architectures (EDAs), known as Event-Driven Service-Oriented Architectures (ED-SOAs or SOAs 2.0) [9]. The latter enable us to establish decoupled communications between users, applications and services. The integration of CEP with SOA 2.0 is required to be able to detect real-time, relevant or critical situations in these complex and heterogeneous systems, as well as for the execution of appropriate actions.

In order to respond to these needs, in this paper, we propose MEdit4CEP, a model-driven solution for real-time decision making in SOA 2.0. The main aim of our proposal is to make domain experts' tasks of defining both

event patterns to be detected and alerts for real time notification easier, hiding all implementation details from them.

More specifically, our solution is composed of a model-driven approach for CEP in SOA 2.0 together with both a graphical modeling editor for CEP domain definition and a graphical modeling editor for event pattern and action definition as well as code generation. The key feature of our event pattern editor is its ability to reconfigure itself for different CEP domains, modeled by domain experts. The fact that the editor can reconfigure the tool palette dynamically from different CEP domain models allows users to enjoy a graphical interface adapted to the specific context required. Our proposal has been evaluated and discussed not only through the development of a case study but also by comparing it to other existing ones. As a result, we can assert that this is a novel solution for bringing CEP technology closer to any user, positively impacting on the decision making process.

The remainder of the paper is organized as follows: Section 2 includes background on Model-Driven Development (MDD), SOA 2.0 and CEP, and describes related work. Section 3 overviews our model-driven approach for CEP in SOA 2.0. Afterwards, Section 4 presents our graphical modeling editors for CEP domain and event pattern definition, and Section 5 describes our proposal for integrating CEP with SOA 2.0. Then, a case study for network security is conducted in Section 6. Finally, our approach and graphical editors are evaluated in Section 7, followed by conclusions and future work discussed in Section 8.

2. Preliminaries

Background information and an overview of related work are provided in this section.

2.1. Background

The subject matters relevant to the scope of this paper, MDD, SOA 2.0 and CEP, are introduced in this subsection.

2.1.1. Model-driven development

MDD is a paradigm in software development that focuses on essential aspects of the system, postponing the choice of the most appropriate technology for implementation [10]. Model definition and transformations between models and from models to code are key factors in developing automatic systems. The high reusability and reliability of the code generated by this software paradigm, as well as its increased productivity and less costly maintenance, have led to its application in various fields [11].

In this scope, a model is a simplified representation of a given reality with the aim of understanding it better. A graphical representation of a model is called a diagram. Models are created by using Domain-Specific Modeling Languages (DSMLs), whose definition consists of three distinct parts: 1) the abstract syntax that consists of both a metamodel –a model describing language concepts and relationships between them– and validation rules to check whether a model is well formed, 2) the concrete syntax or DSML notation –the set of useful graphical symbols for drawing diagrams–, and 3) transformations between models and model to code for software automation.

2.1.2. Event-driven service-oriented architecture

SOA is a logical way of designing a software system with the aim of providing services to end user applications or other services distributed in a network, utilizing published and discoverable interfaces [12].

On the other hand, EDA is a style of architecture in which one or more components in a software system are executed upon receiving event notifications [13]. An event can be defined as anything that happens or could happen [9], but also anything that could happen but does not happen.

These architecture types are not exclusive but complementary, so they can be combined resulting in ED-SOA or SOA 2.0, an evolution of traditional SOA in which communication between users and services occurs by means of events [9]. To achieve this integration, an Enterprise Service Bus (ESB) is an appropriate candidate as middleware because of its functionalities [14]: location transparency, transport protocol conversion, message transformation, message routing, message enhancement, security, and monitoring and management.

2.1.3. Complex event processing

CEP is an emerging technology that allows us to analyze and correlate huge amounts of data in form of events with the aim of detecting relevant or critical situations (complex events) in real time.

A situation is an event occurrence or an event sequence that requires an immediate reaction [13]. A simple event is indivisible and happens at a point in time; a complex event contains more semantic meaning which summarizes a set of other events. Events can be derived from other events by applying or matching event patterns; these are defined by using specific languages developed for this purpose, known as EPLs. A CEP engine is the software used to match these patterns over continuous and heterogeneous event streams, and to raise alerts about complex events created when detecting such event patterns.

As depicted in Figure 1, CEP is performed in 3 stages: 1) event capture –it receives events to be analyzed by CEP technology–, 2) analysis –from the event patterns previously defined in the CEP engine, it will process and correlate the information in the form of events in order to detect critical or relevant situations in real time–, and 3) response –after detecting a concrete situation, it will be notified to the system, software or device in question–.

The main advantage of using this technology to process complex events is that these can be identified and reported in real time, thus reducing latency in decision making, unlike the methods used in traditional software for event analysis. Other relevant advantages are [15]: decision quality improvement, faster and automatic reply, information overload prevention and human workload reduction.

Figure 1. Complex event processing stages.

2.2. Related work

The existing graphical editors for event pattern definition and code generation as well as proposals for integrating CEP with EDA, SOA or SOA 2.0 are described in this subsection.

2.2.1. Graphical editors for event pattern definition and code generation

Currently, there are some editors which define event patterns and transform them into EPL code. Nevertheless, most of them require non-expert CEP users to write by hand at least some of the EPL code that implements the event pattern in question.

Some of these editors are developed by large companies as complementary tools for their event processing systems, such as Oracle CEP Visualizer [6], StreamBase Studio [8] and SAP Sybase ESP Studio [7]. Consequently, they usually perform event pattern transformations just for their own EPL. This dependence

between the CEP engine and the editor may force users to define n times the same event pattern for each of these CEP systems where the event pattern must be detected. What aggravates this situation is the fact that users – rather computer programmers, since they are usually required to write some code by hand– will have to invest a lot of time learning each EPL as well as familiarizing themselves with the use of each editor. On the other hand, the definition of actions to be carried out when detecting event patterns are not often included in these editors and, if so, a similar problem regarding implementation by non-technological users is presented. Precisely, our graphical editors proposed in this paper aim to solve all these mentioned problems.

Other graphical editors have been developed by research projects and groups. Kavelar et al. [16] have created a web application with two main functionalities: 1) the definition of event patterns and actions together with the administration and configuration of the proposed CEP system, and 2) the use of the previously defined event patterns and actions in order to describe situations of interest for a particular domain. This approach has some limitations with respect to our proposal. Firstly, this tool requires CEP experts to be responsible for managing and configuring the CEP system as well as defining event patterns and actions, therefore promoting the dependence of domain experts on CEP experts. Secondly, this definition is given by using natural language, which can lead to defining ambiguous event patterns and actions. Thirdly, functionality and usability levels are lower than the ones provided by our editors, as discussed in Section 7.

Sen et al. [17] have also developed a web editor by using the GWT (Google Web Toolkit) framework [18], which enables event pattern definition from event nodes, operators and action nodes as well as EPL code generation. Afterwards, a new version of this editor renamed as PANTEON has been launched from the FP7 European project ALERT [19]. Although this editor provides advantages over Kavelar's, since domain experts can define event patterns themselves, without help from CEP experts, the editor also has some limitations regarding its functionality and usability levels (see Section 7).

Additionally, the CEP Editor [20] developed by project SocEDA (Social Event-Driven Architecture) has also been built with the same purpose but presents some limitations as discussed in Section 7.

2.2.2. Proposals for integrating CEP with EDA, SOA or SOA 2.0

In order to detect situations of interest in real time, several works integrating CEP technology with EDA, SOA or SOA 2.0 have been proposed over recent years. Table 1 summarizes these proposals indicating the event processing software and middleware used in each case.

Table 1. Proposals for integrating CEP with EDA, SOA or SOA 2.0

CEP with	Proposal	Topic	Event processing software	Middleware
EDA	[21]	Pattern-based architectures for event processing systems	Own	Own
	[22]	Decision support in traffic management systems	Esper	Own
	[23]	Network analysis and surveillance system based on CEP	Esper	Own
SOA	[24]	Smart home environments	Esper	SCA
	[25]	Management of waste water treatment plants	Drools	JBoss ESB
	[26]	Dynamic monitoring of SOA execution environments	Esper	ServiceMix ESB
SOA 2.0	[19]	Coordination among collaborative and distributed virtual teams developing software	ETALIS	Petals ESB
	[27]	High performance ESB platform for CEP	Own	Own
	[28]	Compliance-driven models, languages and architectures for services	Esper	ServiceMix ESB
	[29]	Exhaust gas detection services	Own	Own
	[30]	WSDL and BPEL extensions for EDA	Apache ODE	Non-concrete ESB
	[31]	Web services-based event-driven SOA	MS Workflow Foundation	.NET framework
	[32]	The power of EDA in conjunction with SOA	Non-concrete CEP engine	Non-concrete ESB
	[33]	Intelligent video surveillance systems	Esper	Own
	[34]	Methodology for dynamic service compositions	Esper	Open ESB
	[35]	Method for specification and implementation of applications on a SOA	Esper	BPEL workflows
	[36]	Event processing mechanism in enterprise information systems based on RFID	Own	Own

Regarding proposals integrating CEP with EDA, none of them uses an ESB, which would considerably facilitate such integration.

With regards to the proposals integrating CEP with SOA, all of them describe architectures that differ from our proposal. On the one hand, none of them uses Mule ESB [37], one of the best current ESBs, as maintained by Forrester [38], because of its ability to integrate itself with cloud platforms as well as multiple tools and domain scenarios. On the other, Sottara et al. do not use Esper [39], one of the best-known and most widely used open source CEP engines because of its ability to process thousands of events per second and to dynamically manage events in map format –providing greater flexibility when defining event types with nested properties–.

By analyzing the proposals which combine CEP and SOA 2.0, these have some limitations compared to our proposal. First of all, none of them has integrated Esper engine with Mule ESB, whose combination would provide the aforementioned benefits. Secondly, most of these proposals have no graphical editors to help domain experts to define event patterns with their corresponding actions and to transform them into code. Finally, although some of these architectures have certain similarities with the architecture proposed in this paper

according to their functionalities, they have been designed for a particular domain, such as waste water treatment plants or intelligent video surveillance, whereas our architecture is almost independent of the domain where it needs to be applied.

3. Model-driven approach for CEP in SOA 2.0

We propose a model-driven approach for CEP in SOA 2.0 whose main purpose is the definition of high-level models, which are approachable and understandable to any user. These models will be transformed into code that can be executed in CEP engines and ESBs. Figure 2 sketches this approach, which includes two user roles:

Figure 2. Model-driven approach for CEP in SOA 2.0.

- **Domain expert:** it represents people who have a vast knowledge of a specific area. These experts are ideal candidates to accurately define the concrete domain where CEP can be applied. In other words, they can define which event types along with their properties are necessary for describing all the information about a particular domain, ensuring that the designed models are a true reflection of reality.
- **End user:** it encompasses all those who have the knowledge required to specify the conditions that must be met in order to detect critical or relevant situations for a specific domain, but who are not familiar with any EPL for implementing event patterns. Therefore, these end users –or simply users– will be the ones who use our graphical editors (see Section 4) to graphically define event patterns, hiding them from the implementation information. An end user may, in turn, be a domain expert.

This approach is composed of two distinct parts. The first part covers the process that will take place at design time (left-hand side of Figure 2) whereas the second part includes the process to be executed at runtime (right-hand side of Figure 2). Following the illustration's numbering sequence, the steps involved from the definition of the domain where CEP will be applied until the end user becomes aware of a critical or relevant situation are described below:

1. 1a) When the domain expert requests it, the system will automatically generate a graphical CEP domain model by inferring the event types along with their properties from the event producer information reaching the ESB. Then, the domain expert will be able to make changes to this generated domain model, for example, add its textual description. 1b) Alternatively, the domain expert creates a graphical domain model from scratch, defining event types along with their properties for the domain in question. To carry out this first step, a graphical modeling editor for CEP domain definition has been proposed (see Section 4.1).
2. Once the domain model is defined, the CEP domain editor will be responsible for its validation. If it is not valid, the domain expert will be advised to correct the detected errors. This model will then be saved and can be exported and imported so as to be reused and shared with other experts (see Section 4.1).
3. From the previously defined CEP domain, the end user will create pattern models. In order to do this, a graphical modeling editor for event pattern definition has been proposed (see Section 4.2). Note that this editor can be reconfigured with any of the existing CEP domain models.
4. Once pattern models have been defined, the pattern editor will be responsible for their validation. If any of them is invalid, the user will be asked to correct the detected errors. These models will then be saved and can be also exported and imported so as to be reused and shared with other users (see Section 4.2).
5. Each of these pattern models will be automatically transformed into code, which consists of both the code implementing the conditions that must be met so that the CEP engine can detect critical situations, and code of actions to be performed in the ESB when detecting such situations (see Section 4.2).
6. The generated event pattern code will be added to the CEP engine at runtime (see Section 5).
7. The generated action code will be added to the ESB at runtime (see Section 5).
8. By means of simple events reaching the CEP engine from the ESB and event patterns added at runtime, this engine will create new complex events when detecting such patterns. Afterwards, these complex events will be sent to the ESB, which will be in charge of broadcasting them to every event consumer interested in these events –specified in the actions added to the ESB for each event pattern–, warning

end users about occurred situations (see Section 5). This way, our approach enables real-time decision making in SOA 2.0.

4. Graphical modeling editors

Our graphical modeling editors for CEP domain and event pattern definition are presented in this section.

4.1. Graphical modeling editor for CEP domain

In this subsection, we propose the creation of a graphical editor that supports the modeling of CEP domains, hiding the implementation details necessary to define such domains from domain experts. This editor has been implemented by using GMF (Graphical Modeling Framework) [40] and the Epsilon project [41].

Thanks to this editor, the domain expert will be able to address the execution of steps 1 and 2 required for defining CEP domains in a user-friendly way, according to our model-driven approach (see Section 3). Therefore, the editor would facilitate for any user, expert in a particular domain but not in CEP, the description of event types and properties for the domain, the validation and storage of a CEP domain model as well as its exportation and importation in order to share it with others users.

Figure 3 depicts the built editor for CEP domain definition. This editor mainly has four parts: 1) a tool palette (right panel) from which the domain experts can select the elements to be incorporated into their models, 2) a canvas (central panel) into which users can drag-and-drop event types along with properties for a CEP domain from the palette, 3) a menu (top menu bar) that allows them to easily select the editor action to be executed, 4) a property view (lower panel) for adding or editing information related to the different elements of a designed model, for example, the specific type of an event property.

4.1.1. Editor palette

The editor palette has 2 tools for CEP domain design, as depicted in Table 2. Although *Event* and *EventProperty* have been represented as green icons with letters *E* and *P*, respectively, these icons may be replaced by other images at domain experts' request, indicating the path to the new image. This provides more usability as long as event types and their properties can be easily identified by images of a particular domain.

This editor checks that the user uses the palette correctly; for instance, if he/she tries to drag and drop an event property directly into the canvas, the editor will not allow it since properties must always be added to events.

Figure 3. A screenshot of the graphical modeling editor for CEP domain.

Table 2. CEP domain editor palette tools.

Name	Description	Notation
<i>Event</i>	It describes an event type for a concrete CEP domain.	E
<i>EventProperty</i>	It describes a feature of an event. Nested properties are supported.	P

4.1.2. Options menu

The editor's menu bar is available in any Eclipse application. In addition to menus by default, a *CEP Domain* menu has been added to increase usability. This menu provides the following options: *New* (it creates a new CEP domain), *Auto-detect Domain* (it automatically creates the CEP domain model from events flowing through a SOA 2.0), *Open* (it opens one of the modeled CEP domains), *Save and Validate* (it saves the active domain model and validates it showing encountered problems, if any), *Delete* (it eliminates one of the existing CEP domains). Moreover, two more options have been added to the *File* menu: *Import CEP Domain* (it allows the importation of previously defined CEP domains) and *Export CEP Domain* (it permits exporting CEP domains modeled by the user if previously validated).

4.2. Graphical modeling editor for event pattern

In this subsection, we propose the creation of a graphical editor that supports the modeling of event patterns –the situations to be detected in a particular domain and the actions to be notified to interested users by email or social networking services, among others –. This editor has been implemented using GMF and Epsilon. Notice that an initial version of the metamodel required as the basis for event pattern editor’s creation was published in [42].

Thanks to this editor, users will be able to address the execution of steps 3, 4 and 5, essential for defining patterns in an intuitive way, according to our model-driven approach (see Section 3). Thereby, the editor would provide any user with the description of situations and actions for a domain without knowledge of any concrete EPL or language for implementing actions. Additionally, the editor enables the validation and storage of pattern models, their exportation and importation in order to share them with other users, as well as their transformation into both pattern code to be added to a CEP engine and action code to be deployed into an ESB.

Figure 4 shows the built editor for event pattern definition. This editor has the same four parts as the CEP domain editor: a tool palette, a canvas, a menu and a property view.

4.2.1. Editor palette

The editor palette has 41 tools for designing event patterns, as depicted in Table 3. This palette can be dynamically reconfigured from different CEP domain models, enabling users to enjoy a graphical interface adapted to the specific context required. Specifically, the *Simple Events* category is customized with all event types provided by a modeled CEP domain whereas *Complex Events* is customized with the complex event types defined when designing event patterns for a particular domain. As with the CEP domain editor, this editor allows users to substitute icons with one another and ensures that the tool palette is used correctly. Since the complex event type defined in an event pattern model is automatically incorporated in the palette as a tool in order to reuse it in other pattern models, it will be possible to create an event pattern hierarchy, i.e. a pattern depending on others.

Figure 4. A screenshot of the graphical modeling editor for event pattern.

Table 3. Event pattern editor palette tools.

Category	Name	Description	Notation
	Link	It defines the graphical representation of one or more relationships between operands and operators.	
	Value	It defines a <i>Boolean</i> , <i>Integer</i> , <i>Long</i> , <i>Double</i> , <i>Float</i> or <i>String</i> value.	
Simple Events	Event	It describes an event type for a concrete CEP domain.	
	EventProperty	It describes a feature of an event. Nested properties are supported.	
Complex Events	ComplexEvent	It describes the complex event type to be created upon pattern detection.	
	ComplexEventProperty	It describes a feature of a complex event. Nested properties are supported.	
Pattern Timers	TimerInterval	It waits for the specified time period (<i>years</i> , <i>months</i> , <i>weeks</i> , <i>days</i> , <i>hours</i> , <i>minutes</i> , <i>seconds</i> and <i>milliseconds</i>) before turning to true.	
	TimerSchedule	It turns into true at a defined time (<i>dayOfWeek</i> , <i>dayOfMonth</i> , <i>month</i> , <i>hour</i> , <i>minute</i> , <i>second</i>).	
	WithinTimer	It is permanently evaluated to false if the contained pattern expression does not turn to true during the specified time period (<i>years</i> , <i>months</i> , <i>weeks</i> , <i>days</i> , <i>hours</i> , <i>minutes</i> , <i>seconds</i> and <i>milliseconds</i>).	
Pattern Operators	Every	It selects every event belonging to the specified type.	
	EveryDistinct	It is similar to Every, but eliminates duplicated results according to a given <i>distinct-value</i> expression.	

	<i>FollowedBy</i>	It determines a pattern expression that must be followed by another.
	<i>Range</i>	It specifies the minimum (<i>lowEndpoint</i>) and maximum (<i>highEndpoint</i>) number of times a pattern expression must occur.
	<i>Repeat</i>	It defines how many times (<i>count</i>) a pattern expression must occur.
	<i>Until</i>	It checks a pattern expression until the condition (another pattern expression) is evaluated to true.
	<i>While</i>	It checks a pattern expression while the condition (another pattern expression) is evaluated to true.
<i>Logical Operators</i>	<i>And</i>	It returns a true value only if all operands are true.
	<i>Or</i>	It returns a true value if at least one operand is true.
	<i>Not</i>	It returns a true value if the operand is false, and a false value if the operand is true.
<i>Comparison Operators</i>	<i>Equal</i>	It returns a true value if operand1 = operand2.
	<i>GreaterEqual</i>	It returns a true value if operand1 >= operand2.
	<i>GreaterThan</i>	It returns a true value if operand1 > operand2.
	<i>LessEqual</i>	It returns a true value if operand1 <= operand2.
	<i>LessThan</i>	It returns a true value if operand1 < operand2.
	<i>NotEqual</i>	It returns a true value if operand1 ≠ operand2.
<i>Arithmetic Operators</i>	<i>Addition</i>	It adds two numeric values.
	<i>Division</i>	It divides one numeric value by another.
	<i>Modulus</i>	It returns the remainder of dividing one numeric value by another.
	<i>Multiplication</i>	It multiplies two numeric values.
	<i>Subtraction</i>	It subtracts one numeric value from another.
<i>Aggregation Operators</i>	<i>Avg</i>	It returns the average of the values in an expression.
	<i>Count</i>	It returns the number of values in an expression.
	<i>Max</i>	It returns the highest value in an expression.
	<i>Min</i>	It returns the lowest value in an expression.
	<i>Sum</i>	It adds the values in an expression.
<i>Data Windows</i>	<i>Batching EventInterval</i>	Tumbling window up to the specified number of events (<i>size</i>).
	<i>Batching TimeInterval</i>	Tumbling window that batches events and releases them every specified time period (<i>years, months, weeks, days, hours, minutes, seconds, milliseconds</i>).
	<i>Sliding EventInterval</i>	Sliding window by the specified number of events (<i>size</i>).
	<i>Sliding TimeInterval</i>	Sliding window by the specified time period (<i>years, months, weeks, days, hours, minutes, seconds, milliseconds</i>).
<i>Actions</i>	<i>Email</i>	It indicates an email must be sent with the complex event created when detecting its corresponding event pattern.
	<i>Twitter</i>	It indicates a message must be sent to a Twitter account with the complex event created when detecting its corresponding event pattern.

4.2.2. Options menu

The editor's menu bar is available in any Eclipse application. In addition to menus by default, a *CEP Domain* menu of the domain editor has been added along with a new menu called *Event Pattern: New* (it creates a new

event pattern), *Duplicate Existing Pattern* (it facilitates the event pattern's creation from another one), *Open* (it opens one of the previously modeled event patterns), *Save and Validate* (it saves the active event pattern model and validates it, listing encountered problems, if any), *Generate and Deploy Pattern Code* (it transforms the chosen validated event pattern model into code), *Delete* (it eliminates one event pattern).

By integrating the CEP domain editor with the event pattern editor, users are provided with all functionalities in a single editor. In addition to *Import CEP Domain* and *Export CEP Domain*, two more options have been added to the *File* menu: *Import Event Patterns* (it allows the importation of previously defined event patterns), *Export Event Patterns* (it permits exporting already modeled and validated event patterns).

We would like to point out that, in the current version of this editor, transformation rules have been created to automatically transform pattern models into Esper EPL and XML action code. It is important to highlight that our editor can be extended to support other languages since it allows modeling patterns regardless of the language finally used for their implementation. To this end, new transformation rules per EPL or language for implementing actions which is to be incorporated in the editor, in order to transform pattern models to this new language, must be created.

5. A proposal for integrating CEP with SOA 2.0

In this section, our proposal for integrating CEP with SOA 2.0 is presented and implemented.

5.1. Proposal overview

Steps 1a, 6, 7 and 8 of our model-driven approach are tackled in this proposal. On the one hand, a CEP domain model may be automatically created by inferring event types from events flowing through the architecture. On the other, the even pattern code and action code generated by the pattern editor can be added into the CEP engine and the ESB, respectively, at runtime. Moreover, complex events created by the CEP engine can be sent to the ESB, responsible for alerting end users to these detected situations.

Figure 5 illustrates our proposal for integrating CEP with SOA 2.0. The main building blocks of this architecture type are event producers, event consumers and the ESB.

The event producers generate the events subject to be analyzed by CEP technology in order to detect relevant situations. The event consumers receive the events and trigger concrete types of action. The ESB is the communication channel between event producers and event consumers that allows us to process, transform,

enrich and route messages between components. This is the key element for integrating CEP with SOA 2.0. To successfully integrate it, we have considered the following functionalities to be included in the ESB:

- *Event reception*: it receives raw events generated by event producers. To this end, transport connectors are used to get data into the ESB. These connectors provide adaptation layers for several protocols.
- *Event transformation*: all raw events, regardless of their origin, should share the same format to facilitate the event's processing to the CEP engine. Otherwise, these will be transformed into a common format.
- *Event filtering*: events may be filtered under certain conditions with the purpose of sending only a subset of these to the CEP engine.
- *Domain dynamic generation*: before sending each event to the CEP engine, its type will be analyzed to check if it is already recognized by the engine or if it is a new event type. If the latter is the case, the event type along with property types will be automatically inferred, even if the properties are nested; then these types will be registered into the CEP engine.

Figure 5. Proposal for integrating CEP with SOA 2.0.

- *Event sending to the CEP engine*: once events have been received in the ESB –perhaps also transformed and/or filtered–, these will be sent to the CEP engine at runtime.
- *Event pattern addition to the CEP engine*: the code of the desired event patterns to be detected can be added to the CEP engine at runtime.
- *Action for event pattern addition to the ESB*: the code of the desired actions to be triggered can be added to the ESB at runtime.
- *Event pattern detection*: if one of the registered event patterns is fulfilled, then a complex event summarizing the occurred situation will be created and sent to the ESB.
- *Complex event reception*: the ESB will receive complex events as they are created from pattern detection.
- *Decision making (actions for detected complex events)*: the ESB will send complex events to the event consumers subscribed to these event types. For example, complex events may be notified to users by email or a Twitter account; in addition, these may also be stored in an event repository for retrospective processing.

Note that *Event reception*, *Event transformation* and *Event filtering* functionalities are dependent on particular event producers and, therefore, on the scenario where SOA 2.0 is applied.

5.2. Proposal's implementation

The implementation of our proposal for integrating CEP with SOA 2.0, as well as the connection between this proposal and our graphical modeling editors are described below.

A Mule ESB application with 5 flows –the main control structure for carrying out the whole process and management of message flowing through this ESB– have been implemented as follows:

- *EventReceptionAndManagement-DomainI*: this flow contains the described *Event reception*, *Event transformation* and *Event filtering* functionalities. It has some endpoints for data reception from event producers. Then, a transformer converts this heterogeneous data into a common format for events – *Map<String, Object>*, the string indicating the event name and the object stores event properties along with their values–. Optionally, an event filter may be used. Finally, these events are sent to the following flow. Notice that this first flow is the only one dependent on the domain where SOA 2.0 is applied, depending on the event producer types with which it is connected. This enables the creation of

analogous flows for other domains, such as *-Domain2* and *-Domain3*, without the need of any modification of the other 4 flows in our implemented architecture.

- *DomainDynamicGenerationAndEventSendingToEsper*: this flow implements the *Domain dynamic generation* and *Event sending to the CEP engine* functionalities. In particular, it continuously checks for new event types, in which case these will be registered into the Esper engine, and sends all received events to this engine. In addition, every new inferred event type will be read by the CEP domain editor to automatically create the CEP domain model –or modify it if the model already exists–. Then, the user may modify this model, for example, by adding a textual description.
- *EventPatternAdditionToEsper*: this flow is responsible for adding an event pattern to the CEP engine at runtime when a file containing EPL code is generated automatically by the event pattern editor.
- *ActionForEventPatternAdditionToMule*: this flow adds actions for an event pattern in the ESB at runtime, when a file containing XML action code is generated automatically by the event pattern editor.
- *ComplexEventReceptionAndDecisionMaking*: this flow includes the *Event pattern detection*, *Complex event reception* and *Decision making* functionalities. Specifically, it receives complex events created by the Esper engine when detecting already registered patterns. Then, decisions will be automatically performed, executing all actions previously specified for the event pattern in question. This way, complex events will only be notified to consumers who request it.

6. Network security case study

Computer networks are critical elements of current Information Technology (IT) infrastructures since faults or attacks on these networks can have large impacts, such as huge financial losses or threats to human life [43]. For this reason, analyzing and monitoring networks as well as the traffic in them is essential to prevent, or at least promptly detect, such critical situations.

Packet capturing is one of the existing techniques for gathering traffic information in TCP/IP networks. The packets transmitted in a computer network are captured by a packet analyzer or sniffer. There are two main challenges to be faced [43]: the huge amount of data to be continuously managed and the need to process it as soon as possible. With the aim of mitigating these problems, Gad et al. propose the use of CEP technology. Thanks to event pattern definition, critical situations in computer networks can be detected in real time, such as congestion situations or denial of service attacks.

In this case study, we apply our model-driven solution for real-time decision making in SOA 2.0 to a network security scenario. The event producer used in this architecture is an event generator (sniffer) developed by Gad, a member of the IT Security, Network Security and Privacy Research Group at Frankfurt University of Applied Sciences (Germany). The email service works as an event consumer.

In the next subsections, the new Mule flow for event reception and management in this network domain is described, as well as the steps to be followed by the domain expert –according to our model-driven approach– from the network domain definition until the user can receive real-time notifications about occurred situations.

6.1. Event reception and management

As previously mentioned, our proposal for integrating CEP with SOA 2.0 (see Section 5) is extensible to different application domains, only requiring the implementation of a new Mule flow in order to connect the architecture with the concrete event producer. For this reason, the *EventReceptionAndManagement-NetworkAnalysis* flow has been implemented to integrate Mule with the sniffer to be used as event producer in this case study.

This flow is composed of a JMS endpoint responsible for establishing communication between the event generator and Mule, a collection splitter for separating data obtained from the sniffer into simple events (type *sniffer.header.parsed*) and a flow reference to send each simple event to the flow called *DomainDynamicGenerationAndEventSendingToEsper*, which will automatically infer the CEP domain and send these events to the CEP engine.

6.2. CEP domain model definition

The first step of our model-driven approach is the CEP domain definition by the domain expert. To cope with this task, two options are possible: a) automatically obtaining the CEP domain graphical model from inferring the types of event flowing the SOA 2.0 or b) manually designing the CEP domain graphical model by using the CEP domain editor or even the event pattern editor.

We have chosen the automatic generation of the CEP domain model for this case study (see Figure 3). As a result, this domain contains the *sniffer.header.parsed* event type along with its event properties: *ts* (timestamp for captured packet), *ipSrc* (IP source address), *ipDst* (IP destination address) and *icmpType* (ICMP type: *echo request* or *echo reply*), among others. Moreover, the domain name has been set to *network-analysis*, a textual

description has been indicated for the domain and a customized icon has been assigned to the event type, improving usability.

6.3. CEP domain model validation and storing

Once designed the network analysis domain, it has been automatically validated and stored. Then, the event pattern editor has been automatically reconfigured for this domain, i.e. the *sniffer.header.parsed* event type has been added as a tool in the *Simple Events* category of the editor palette. This means that end users do not have to worry about how to define event types and their properties, since these are graphically represented when dragging and dropping the tool. In addition, they cannot modify given event types, avoiding the creation of incorrect event patterns for the same domain.

6.4. Event pattern model definition

By using the event pattern editor reconfigured for the network analysis domain, the *icmp echo request*, *icmp echo reply* and *icmp ping response time* event patterns have been modeled. These are detailed below:

6.4.1. ICMP echo request pattern definition

- **Name:** *icmp_echo_request*.
- **Description:** this pattern allows us to detect a control message sent to a host with the purpose of receiving an *echo-reply* message from it.
- **Pattern condition:** the value of the *icmpType* property of a *sniffer.header.parsed* event must be equal to the *echo request* value.
- **New complex event:** *icmp_echo_request*. A new complex event will be created with the following properties: *timestamp* (the timestamp in nanoseconds –the *ts* property– of the *sniffer.header.parsed* event), *source* (its value must equal the *ipSrc* property of the *sniffer.header.parsed* event) and *destination* (its value must equal the *ipDst* property of the *sniffer.header.parsed* event).
- **Action:** *email*. Each new *icmp_echo_request* complex event, which is created when the event pattern condition is satisfied, will be sent to the email address(es) specified in the *to* attribute in the *Email* component. To that end, the *from* address, host, port, username and password, as well as the subject – for example, *Alert: ICMP Echo Request* –, must be indicated.

Figure 6 shows the *icmp_echo_request* event pattern modeled using the pattern editor. As shown, a customized icon has been assigned to the complex event type. Notice that the *sniffer.header.parsed* event is not completely shown due to space restrictions.

Figure 6. ICMP echo request event pattern modeled using the pattern editor.

6.4.2. ICMP echo reply pattern definition

- **Name:** *icmp_echo_reply*.
- **Description:** this pattern allows us to detect a control message generated as a response to an *echo-request* message.
- **Pattern condition:** the value of the *icmpType* property of a *sniffer.header.parsed* event must be equal to the *echo reply* value.
- **New complex event:** *icmp_echo_reply*. A new complex event will be created with the following properties: *timestamp* (the timestamp in nanoseconds –the *ts* property– of the *sniffer.header.parsed* event), *source* (its value must equal the *ipSrc* property of the *sniffer.header.parsed* event) and *destination* (its value must equal the *ipDst* property of the *sniffer.header.parsed* event).
- **Action:** *email*. Each new *icmp_echo_reply* complex event will be sent to the email address(es) specified in the *to* attribute in the *Email* component.

6.4.3. ICMP ping response time pattern definition

- **Name:** *icmp_ping_response_time*.
- **Description:** this pattern allows us to calculate the temporal difference between the occurrence of an *ICMP echo request* message and an *ICMP echo reply* message.
- **Pattern conditions:**
 - Every *icmp_echo_request* complex event –generated by the *icmp_echo_request* pattern– is followed by an *icmp_echo_reply* complex event, generated by the *icmp_echo_reply* pattern.
 - Additionally, the value of the *source* property of the *icmp_echo_reply* complex event is equal to the *destination* property of the *icmp_echo_request* complex event, and the value of the *destination* property of the *icmp_echo_reply* event is equal to the *source* property of the *icmp_echo_request* event.
- **New complex event:** *icmp_ping_response_time*. A new complex event will be created with the following properties: *requestTimestamp* (the value of the *timestamp* property of the *icmp_echo_request* complex event), *responseTimestamp* (the value of the *timestamp* property of the *icmp_echo_reply* complex event), *time* (the timestamp difference between the *icmp_echo_reply* and *icmp_echo_request* complex events), *source* (the value of the *source* property of the *icmp_echo_request* complex event) and *destination* (the value of the *destination* property of the *icmp_echo_request* complex event).
- **Action:** *email*. Each new *icmp_ping_response_time* complex event will be sent to the email address(es) specified in the *to* attribute in the *Email* component.

Figure 4 illustrates the *icmp_ping_response_time* pattern modeled using the event pattern editor.

6.5. Event pattern model validation and storing

Once each pattern has been designed, it has to be automatically validated and stored after checking that the model is correct. Afterwards, the event pattern editor has been automatically reconfigured adding every new complex event type as a tool in the *Complex Events* category of the editor palette (see Figure 4). This allows end users to be able to graphically define an event pattern hierarchy by dragging and dropping previously added tools from the *Complex Events* category into a new event pattern model. For instance, the *icmp_ping_response_time* pattern has been designed thanks to the use of the *icmp_echo_request* and *icmp_echo_reply* tools that were automatically added to the *Complex Events* category after modeling the *icmp_echo_request* and *icmp_echo_reply* patterns.

6.6. Event pattern model transformation to code

After designing, validating and storing the event patterns, both the EPL code to be added in the Esper engine at runtime as well as the XML code for actions to be carried out in Mule can be automatically generated by the event pattern editor. As a result, a file *.epl* will be created per event pattern and, if the pattern contains actions, another file *.action*. In order to illustrate the generated files we show those generated for the ICMP echo request pattern in the following lines.

The implementation of the ICMP echo request pattern (see Section 6.4.1) in EPL is shown in Listing 1. First of all, the *from* clause indicates for which event type *–sniffer.header.parsed–* the pattern condition must be evaluated, assigning the *a1* alias to this type. The condition *–the value of the icmpType property of a sniffer.header.parsed event must be equal to the echo request value–* is specified by using the *where* clause. Next, event properties to be selected from the events matching the pattern condition are specified: timestamp, IP source and IP destination. Then, a complex event along with its properties *–icmp_echo_request(timestamp, source, destination)–* is created and inserted into a new Esper event flow named exactly as the complex event, namely *icmp_echo_request*.

Listing 1. File *icmp_echo_request.epl* generated by the event pattern editor.

```
@Name('icmp_echo_request')
insert into icmp_echo_request
select a1.ts as timestamp,
      a1.ipSrc as source,
      a1.ipDst as destination
from sniffer.header.parsed a1
where a1.icmpType = 'echo request'
```

Listing 2. File *icmp_echo_request.action* generated by the event pattern editor.

```
<?xml version="1.0" encoding="UTF-8"?>

<mule xmlns: [...] xsi:schemaLocation= [...] >

  <flow name="icmp_echo_request">
    <processor-chain>
      <set-payload value="Detected Alert '[message.inboundProperties[
'eventPatternName']]': #[payload]" doc:name="Set Email Body"/>
      <smtps:outbound-endpoint host="smtp.uca.es" port="465" user="*****"
password="*****" to="juan.boubeta@uca.es" from="juan.boubeta@uca.es" cc=""
subject="Alert: icmp echo request" responseTimeout="3000" doc:name="SMTP"/>
    </processor-chain>
  </flow>
</mule>
```

In addition, the implementation of the action for this pattern in XML is shown in Listing 2. The first line of this file is the XML declaration followed by the Mule namespace and XML schema declarations. Next, a new Mule flow named equal to its corresponding event pattern *–icmp_echo_request–* is presented. This flow is composed

of both a *set payload* transformer for setting the concrete format for the email payload to be sent –containing the complex event received in the Mule flow called *ComplexEventReceptionAndDecisionMaking*– and a SMTP connector that makes sending complex events (alerts) as emails possible.

6.7. Critical situation detection and notification

In order to detect and notify the graphically modeled security situations, the EPL event pattern code generated by the editor is automatically registered in the Esper engine, and also the XML action code is added to the Mule ESB at runtime. Then, the *ComplexEventReceptionAndDecisionMaking* flow will receive complex events and make decisions by executing actions for the patterns responsible for creating these events; in this case, sending notifications by email to interested users.

7. Evaluation and discussion

In this section, we evaluate and discuss our graphical editors for modeling CEP domains and event patterns as well as our model-driven approach for real-time decision making in SOA 2.0.

7.1. Graphical editor evaluation

A functionality and usability evaluation of our graphical editors as well as a comparison with other existing editors is conducted below.

7.1.1. Functionality and usability

The evaluation of both the graphical editor for CEP domain modeling and the reconfigurable graphical editor for event pattern modeling and code generation has been conducted through a questionnaire provided to end users. Since the reconfigurable graphical editor itself included the graphical editor for CEP domains –therefore including all the functionality provided by the latter–, the evaluation has been focused on the event pattern editor in order to avoid overloading respondents.

The questionnaire designed to evaluate the editor’s usability and functionality is based on the one proposed in [44], which is used for evaluating an event pattern editor called PANTEON and developed in the FP7 European project ALERT. Therefore, the validity of our questionnaire is supported by the European one. It is necessary to highlight that our editor provides significant advantages over PANTEON as demonstrated in Section 7.1.2.

Our questionnaire consists of 19 questions to assess the editor’s functionality and usability as follows:

- Q1: do you consider yourself an expert in complex event processing? (No; Yes)

- Q2: are you an expert on the domain for which you are going to define event patterns? (No; Yes)
- Q3: how would you like to define event patterns? (Using a graphical editor; Coding the pattern using a programming language; Both using a graphical editor and coding the pattern using a programming language)
- Q4: what type of user is most suitable to use the editor? (Domain experts –programming knowledge not required–; Programmers; I do not know; Others)
- Q5: is it clear what the purpose of the editor is? (No; A little; Moderately; A lot; Completely)
- Q6: has the editor satisfied your expectations in general? (No; A little; Moderately; A lot; Completely)
- Q7: what are the functionalities provided by the editor for CEP domains? [multiple answers are allowed] (To define a new domain graphically; To auto-detect a domain; To validate and save a domain model; To modify and delete an existing domain model; To customize a domain with images; To import and export a domain model; Others)
- Q8: what are the functionalities provided by the editor for event patterns? [multiple answers are allowed] (To graphically define new event patterns; To save time; To generate code for modeled event patterns; To validate and save modeled event patterns; To duplicate, modify and delete existing modeled event patterns; To customize event patterns with images; To import and export the modeled event patterns; To notify complex events –relevant or critical situations– to interested users, for example, through email or social networking services.)
- Q9: do you consider the functionalities provided by the editor useful? (No; Yes)
- Q10: do you consider any functionalities provided by the editor irrelevant? (No; Yes)
- Q11: do you consider any tools of the editor palette irrelevant? (No; Yes)
- Q12: have you been able to successfully create the required event patterns for a particular domain? (No; A little; Moderately; A lot; Completely)
- Q13: once the event patterns are graphically defined, is it clear what the purpose of these patterns is? (No; A little; Moderately; A lot; Completely)
- Q14: what EPL skill level do you consider should be required for defining graphical event patterns? (Novice; Beginner; Competent; Advanced; Expert)

- Q15: do you think that the editor could reduce the time needed to define event patterns, instead of manually implementing them by using a concrete EPL? (No; A little; Moderately; A lot; Completely)
- Q16: have you been able to easily find each option in the editor for modeling event patterns? (No; A little; Moderately; A lot; Completely)
- Q17: have you been able to easily understand the given task to be done by using the editor? (No; A little; Moderately; A lot; Completely)
- Q18: how long have you needed to perform the given task? (in minutes)
- Q19: do you have any suggestions for improving the editor? (No; Yes)

The evaluation has been completed by 35 experts in computer security from the University of Cádiz (Spain) and the University of Applied Sciences Frankfurt am Main (Germany). They have been asked to graphically define the three event patterns described in the case study (see Section 6). To do this, a textual description for each event pattern and the network analysis domain automatically detected by the CEP domain editor have been provided to these domain experts. Then, they have had to import this domain in the event pattern editor before starting modeling such patterns.

The respondents have been classified into two groups: experts in security domain but not in CEP (group 1) and experts in security domain and also in CEP (group 2). The results from the questionnaire are shown in Table 4.

One of the highlighted results is that 88.6% of those polled prefer to define event patterns by using a graphical editor as well as implementing them, while 11.4% prefer programming patterns by hand without using a graphical editor. This justifies the need to develop an editor for event pattern definition intuitively.

In addition, 97.1% of respondents consider that domain experts, who may not have programming skills, would benefit most from the editor. In fact, this is one of its main purposes: abstracting the implementation details from any novice user in the required technologies.

Whilst Group 2 knows perfectly well what the purpose of the editor is, as they are experts in CEP, this is not the case for Group 1. This might be because they have not been provided with any user manuals before doing the evaluation with the sole intention of checking whether all menu options and tools are presented by the editor in a user-friendly way. We firmly believe that users' experiences with the editor will improve significantly when a manual is provided.

The editor has satisfied the expectations of the majority of respondents and all of them have been able to recognize all the functionalities provided by the editor, considering them really useful. Furthermore, none of them have considered any editor functionalities nor palette tools irrelevant.

Table 4. Results obtained from the graphical editor evaluation questionnaire.

Question	Answer	Group 1 (%)	Group 2 (%)	Total (%)
Q1	No	100	0	71.4
	Yes	0	100	28.6
Q2	Yes	100	100	100
Q3	Using a graphical editor	68	20	54.3
	Coding the pattern using a programming language	8	20	11.4
	Both using a graphical editor and coding the pattern using a programming language	24	60	34.3
Q4	Domain experts (programming expertise not required)	96	100	97.1
	Programmers	4	0	2.9
Q5	Moderately	24	0	17.1
	A lot	56	0	40
	Completely	20	100	42.9
Q6	Moderately	12	0	8.6
	A lot	64	40	57.1
	Completely	24	60	34.3
Q7	All functionalities	100	100	100
Q8	All functionalities	100	100	100
Q9	Yes	100	100	100
Q10	No	100	100	100
Q11	No	100	100	100
Q12	Moderately	4	0	2.9
	A lot	12	0	8.6
	Completely	84	100	88.6
Q13	Moderately	16	0	11.4
	A lot	60	0	42.9
	Completely	24	100	45.7
Q14	Novice	12	20	14.3
	Beginner	76	80	77.1
	Competent	12	0	8.6
Q15	Moderately	8	0	5.7
	A lot	40	70	48.6
	Completely	52	30	45.7
Q16	A little	8	0	5.7
	Moderately	16	0	11.4
	A lot	48	20	40
	Completely	28	80	42.9
Q17	Moderately	8	0	5.7
	A lot	52	10	40
	Completely	40	90	54.3
Q18	10 – 20 min.	8	30	14.3
	20 – 30 min.	32	60	40
	30 – 40 min.	56	10	42.9
	40 – 50 min.	4	0	2.9
Q19	No	76	70	74.3
	Yes	24	30	25.7

Regarding the satisfaction of creating the required patterns for the network security case study, the vast majority of users created them successfully. This demonstrates that the editor has a good degree of usability and functionality since non-experts in CEP have been able to do it without a user manual. Using a manual would probably improve these results.

Once the event patterns have been graphically defined, their purpose is clear for all Group 2 users but not for Group 1 ones, since the latter did not have previous knowledge of CEP technology or its usage before starting the evaluation.

Moreover, 77.1% of all respondents believe that the required EPLs skill level to define graphical event patterns should be beginner, compared to 14.3% who chose novice users and to 8.6% who indicated competent users. This underlines the previously formulated hypothesis: simple training on basic CEP concepts like *complex events* and *event patterns*, excluding implementation details, would increase user experiences with the editor. Not providing such training in basic CEP concepts may be the reason why some users have not been able to easily find each option in the editor for modeling event patterns, since they are unfamiliar with, for example, the use of concrete pattern operators, such as *Every* or *FollowedBy*.

Most users have been able to easily understand the given task and believe that the graphical editor could notably reduce the time needed to define event patterns, instead of implementing them manually by using a particular EPL. Notice that the required time for both familiarization with the editor and defining the three event patterns was around 30 minutes –lower for CEP experts–. From our experience, the time required by anyone, who neither knows CEP nor any EPLs, for implementing the correct syntax of an event pattern is much higher than the required time when using our graphical editor; even more so if the end user is not a computer engineer.

Since our solution also generates the XML code for the actions to be deployed in SOA 2.0, not using our editor will imply that end users have to know both the XML language to implement the actions by hand and other technologies and languages for integrating the CEP engine with SOA 2.0. Undoubtedly, this will be impossible for users who are experts in the domain but not in the technologies.

Finally, respondents made some suggestions that have been incorporated to the latest version of the editor.

7.1.2. A comparative study with other existing editors

We have conducted a comparative study of our model-driven solution (MEdit4CEP) with other existing editors, considered by us as potential competitors based on their functionality and usability: PANTEON and the CEP

Editor [20], developed by the project SocEDA. Table 5 summarizes the results of the study in which our solution is compared to the other editors, according to 28 functionality and usability criteria.

Table 5. Comparative study of our model-driven solution with other existing editors.

No.	Criteria	MEdit4CEP	PANTEON	CEP Editor
1	Eclipse-based graphical editor	X		
2	GMF-based graphical editor	X		
3	Integration with a CEP engine	X	X	X
4	Integration with an ESB	X	X	X
5	CEP domain reconfigurable editor	X		
6	Editor palette customization at runtime	X	X	
7	CEP domain graphical definition	X		
8	CEP domain auto-detection and automatic graphical representation	X		
9	Definition of events with nested properties	X		X
10	CEP domain graphical model validation	X		
11	CEP domain storing	X	X	X
12	CEP domain customization with images	X		
13	CEP domain importation and exportation	X		X
14	Event pattern graphical definition	X	X	X
15	Event pattern action graphical definition	X	X	
16	Availability of numerous operators, operands and data windows	X		
17	Elements represented as graphical nodes	X	X	X
18	Intuitive definition of complex event hierarchy	X	X	
19	Event pattern graphical model validation	X		
20	Event pattern storing	X	X	X
21	Event pattern transformation to code	X	X	X
22	Event pattern customization with images	X		
23	Event pattern importation and exportation	X		X
24	Event pattern deployment into the CEP engine at runtime	X	X	X
25	Event pattern action deployment into the ESB at runtime	X	X	X
26	Complex event notification through several services	X	X	X
27	Adequate level of usability	X		
28	Semi-automatic event pattern recommendation		X	

These results reveal that our solution presents a higher degree of functionality and usability than the other ones.

Regarding the functionalities provided by other editors, the semi-automatic event pattern recommendation has not been provided by our editor yet, which would allow users to create new patterns reusing others recommended by the editor itself.

With regards to usability, we firmly believe that our editor is more user-friendly and intuitive than the other ones. More specifically, PANTEON represents the event patterns as interconnected window nodes composed of buttons and dropdowns, which are used by users to add event properties together with their conditions. One of the main limitations of this representation is that it may cause the design of vast and awkward event patterns. On the other hand, CEP Editor uses interconnected graphical nodes for event pattern definition; however, event

pattern conditions must be implemented using Esper EPL in the *query* node. This is undoubtedly an impediment for non-expert users in CEP.

7.2. Model-driven approach evaluation

Our model-driven approach for CEP in SOA 2.0 is evaluated according to its productivity. To do this, we have measured the Lines Of Code (LOCs) –excluding comments and blank lines– automatically generated and manually implemented for the security case study, as shown in Table 6.

Table 6. LOCs automatically generated and manually implemented for the security case study.

	Automatically			Total	Manually Total
	XMI	EPL	XML		
	Model	Diagram			
Event reception and management flow	0	0	0	5	0
CEP domain	31	356	0	387	0
Event pattern <i>icmp echo request</i>	45	472	7	546	0
Event pattern <i>icmp echo reply</i>	45	472	7	546	0
Event pattern <i>icmp ping response time</i>	46	385	8	464	0
Complex events	20	83	0	103	0

The automatically generated code has been classified according to the programming language in which it has been implemented. In particular, the XMI code includes LOCs of both EMF (Eclipse Modeling Framework) models (column 2 in the table) and their corresponding diagrams (column 3 in the table) automatically generated from modeled CEP domain and event patterns. In addition, an EMF model together with its corresponding diagram are internally generated for the complex event type defined in each of these patterns –necessary for the dynamic customization of the editor palette–. Whereas column 4 shows LOCs of the EPL code automatically generated per modeled pattern to be inserted into the Esper engine, column 5 indicates LOCs of the XML code generated, implementing the event reception and management Mule flow for the case study as well as the actions for each of these patterns to be deployed in the ESB. Notice that the XML code implementing this Mule flow has been generated by Anypoint Studio [45], an Eclipse-based IDE for developing, debugging and deploying Mule applications that provides both a drag-and-drop graphical editor and a XML editor.

Specifically, both the EPL code and XML code generated by the event pattern editor are totally executable by Esper engine and Mule ESB, respectively. Moreover, this code is error-free since the graphical model validation is effected before transforming event patterns. Obviously, the model’s validation and transformation processes have been previously tested. Thanks to model-driven techniques, we have avoided the learning curve in CEP and

SOA 2.0 required for users to be able to implement and deploy error-free event patterns together with their actions.

In summary, all necessary code to enable real-time decision making in SOA 2.0 for the security case study has been generated fully automatically. However, it is dependent on the particular application domain since the event reception and management Mule flow might require manual implementation in order to transform the information from other event producers into events to be sent to the Esper engine. In any case, this is insignificant compared to all automatically generated code.

8. Conclusions and future work

In this paper, we have proposed and implemented a model-driven solution that enables the integration of CEP with SOA 2.0, thus preventing end users from having to learn the technologies required for the detection of relevant or critical situations and decision-making in real time.

In particular, a CEP domain can be modeled at design time. This model can be manually created by a domain expert using our CEP domain graphical editor described in this paper, or it can be automatically inferred from the events flowing through the SOA 2.0 to which this editor is connected. From the modeled CEP domain, users will be able to model patterns about situations of interest to be detected as well as actions to be performed by using the event pattern graphical editor also unveiled in this paper. Subsequently, these event pattern models will be automatically validated and transformed into code, which will be deployed in a CEP engine and an ESB at runtime, making these tasks transparent to users thanks to the use of model-driven techniques.

Regarding the advantages provided by our CEP domain editor, it supports the unification of CEP domain description (event types and properties) by using models, hiding the implementation details necessary to define such domains from domain experts. Additionally, these models can be shared and reused by other domain experts.

Besides, one of the most relevant contributions offered by our event pattern editor is the possibility of modeling event patterns regardless of their implementation. Thanks to the use of model-driven techniques, every event pattern is graphically designed once and it can then be automatically transformed into any specific error-free EPL code, such as Esper EPL, StreamSQL or CCL, as well as into any error-free action code to be executed, such as XML. The fact that the editor is able to reconfigure the tool palette dynamically from different CEP domain models will enable users to enjoy a graphical interface adapted to the specific context required.

As a result, we conclude that our model-driven solution facilitates for any user, neither a programmer nor CEP expert, the graphical definition of situations of interest for a specific domain and their detection from real-time information flowing through complex and heterogeneous systems.

Concerning our future work, we consider a relevant issue to extend the editor with the ability to recommend event patterns in a semi-automatic way. This will allow end users to create new patterns by reusing other event patterns, which will be suggested by the editor itself from the analysis and interpretation of existing pattern usage statistics. Another research work to be developed in the future is the dynamic generation of event patterns from the information in the form of events, which flow through the organization's information systems. This fact would enable users not to have to design event patterns from scratch, but simply make some modifications on them, saving time when modeling event patterns, particularly when a large number of them have to be defined for an application domain.

So far, our model-driven solution has only been evaluated in the research and academic world. Also as future work, we intend to apply it in the industry with the purpose of checking business experts' acceptance and satisfaction levels. Given the scalability and independence of the CEP domain in which the proposed approach is applied, as well as the functionality and usability of our graphical editors, the industry is expected to welcome this research.

Acknowledgements

This work was funded by the Spanish Ministry of Science and Innovation under the National Program for Research, Development and Innovation, project MoD-SOA (TIN2011-27242). The first author thanks Dr. Kappes and researchers Gad and Müller-Bady for the help and hospitality received at Frankfurt University of Applied Sciences when visiting them, where part of this work was evaluated.

References

- [1] J. A. Morente-Molinera, I. J. Pérez, M. R. Ureña, and E. Herrera-Viedma, "On Multi-granular Fuzzy Linguistic Modeling in Group Decision Making Problems: A Systematic Review and Future Trends," *Knowledge-Based Systems*, vol. 74, pp. 49–60, Jan. 2015.
- [2] N. Leavitt, "Complex-Event Processing Poised for Growth," *Computer*, vol. 42, no. 4, pp. 17–20, 2009.
- [3] S. Alonso, F. J. Cabrerizo, F. Chiclana, F. Herrera, and E. Herrera-Viedma, "An Interactive Decision Support System Based on Consistency Criteria," *Journal of Multiple-Valued Logic and Soft Computing*, vol. 14, no. 3–5, pp. 371–385, 2008.
- [4] S. Alonso, E. Herrera-Viedma, F. Chiclana, and F. Herrera, "A Web Based Consensus Support System for Group Decision Making Problems and Incomplete Preferences," *Information Sciences*, vol. 180, no. 23, pp. 4477–4495, 2010.
- [5] D. Luckham, *The Power of Events: An Introduction to Complex Event Processing in Distributed Enterprise Systems*. MA, USA: Addison-Wesley, 2002.

- [6] Oracle, “Oracle Event Processing,” 2015. [Online]. Available: <http://www.oracle.com/technetwork/middleware/complex-event-processing/overview/index.html>. [Accessed: 03-Feb-2015].
- [7] Sybase, “SAP Sybase Event Stream Processor,” 2015. [Online]. Available: <http://www.sap.com/uk/pc/tech/database/software/sybase-complex-event-processing/index.html>. [Accessed: 20-Feb-2015].
- [8] TIBCO, “StreamBase Studio,” 2015. [Online]. Available: <http://www.streambase.com/products/streambasecep/streambase-studio/>. [Accessed: 15-Feb-2015].
- [9] D. Luckham, *Event Processing for Business: Organizing the Real-Time Enterprise*. New Jersey, USA: Wiley, 2012.
- [10] G. Ortiz, “Integrating Extra-Functional Properties in Model-Driven Web Service Development,” 12-Apr-2007. [Online]. Available: <http://rodin.uca.es/xmlui/handle/10498/16036>. [Accessed: 25-Feb-2015].
- [11] T. Stahl, M. Voelter, and K. Czarnecki, *Model-Driven Software Development: Technology, Engineering, Management*. John Wiley & Sons, 2006.
- [12] M. Papazoglou, *Web Services and SOA: Principles and Technology*. Essex, England; New York: Pearson Education, 2012.
- [13] O. Etzion and P. Niblett, *Event Processing in Action*. Stamford, USA: Manning, 2011.
- [14] T. Rademakers and J. Dirksen, *Open-Source ESBs in Action*. Greenwich: Manning, 2009.
- [15] K. M. Chandy and W. R. Schulte, *Event Processing: Designing IT Systems for Agile Companies*. USA: McGraw-Hill, 2010.
- [16] A. Kavelar, H. Obweger, J. Schiefer, and M. Suntinger, “Web-Based Decision Making for Complex Event Processing Systems,” in *IEEE Congress on Services*, Los Alamitos, CA, USA, 2010, pp. 453–458.
- [17] S. Sen, N. Stojanovic, and R. Lin, “A Graphical Editor for Complex Event Pattern Generation,” in *Proceedings of the Third ACM International Conference on Distributed Event-Based Systems*, New York, NY, USA, 2009, pp. 1–2.
- [18] Google, “Google Web Toolkit,” 2015. [Online]. Available: <http://www.gwtproject.org/>. [Accessed: 15-Feb-2015].
- [19] ALERT, “Active support and real-time coordination based on Event pRocessing in FLOSS development,” 2013. [Online]. Available: <http://www.alert-project.eu/>. [Accessed: 17-Feb-2015].
- [20] SocEDA, “CEP Editor,” 2013. [Online]. Available: <http://research.petalslink.org/display/soceda/CEP+Editor>. [Accessed: 10-Feb-2015].
- [21] R. Bruns and J. Dunkel, “Towards Pattern-based Architectures for Event Processing Systems,” *Software: Practice and Experience*, May 2013.
- [22] J. Dunkel, A. Fernández, R. Ortiz, and S. Ossowski, “Event-driven Architecture for Decision Support in Traffic Management Systems,” *Expert Systems with Applications*, vol. 38, no. 6, pp. 6530–6539, Jun. 2011.
- [23] R. Gad, M. Kappes, J. Boubeta-Puig, and I. Medina-Bulo, “Employing the CEP Paradigm for Network Analysis and Surveillance,” in *Proceedings of The Ninth Advanced International Conference on Telecommunications*, Rome, Italy, 2013, pp. 204–210.
- [24] D. Romero, G. Hermosillo, A. Taherkordi, R. Nzekwa, R. Rouvoy, and F. Eliassen, “The DigiHome Service-Oriented Platform,” *Software: Practice and Experience*, vol. 43, no. 10, pp. 1205–1218, Oct. 2011.
- [25] D. Sottara, A. Manservigi, P. Mello, G. Colombini, and L. Luccarini, “A CEP-based SOA for the management of WasteWater Treatment Plants,” in *IEEE Workshop on Environmental, Energy, and Structural Monitoring Systems (EESMS)*, Crema, Italy, 2009, pp. 58–65.
- [26] D. Zmuda, M. Psiuk, and K. Zielinski, “Dynamic Monitoring Framework for the SOA Execution Environment,” *Procedia Computer Science*, vol. 1, no. 1, pp. 125–133, May 2010.
- [27] D. Bo, D. Kun, and Z. Xiaoyi, “A High Performance Enterprise Service Bus Platform for Complex Event Processing,” in *Seventh International Conference on Grid and Cooperative Computing (GCC)*, 2008, pp. 577–582.
- [28] COMPAS, “Compliance-driven Models, Languages, and Architectures for Services,” 2011. [Online]. Available: <http://cordis.europa.eu/fp7/ict/ssai/docs/finalreport-compas.pdf>. [Accessed: 27-Feb-2015].
- [29] M. He, Z. Zheng, G. Xue, and X. Du, “Event Driven RFID Based Exhaust Gas Detection Services Oriented System Research,” in *4th International Conference on Wireless Communications, Networking and Mobile Computing (WiCOM)*, Dalian, China, 2008, pp. 1–4.
- [30] M. B. Juric, “WSDL and BPEL extensions for Event Driven Architecture,” *Information and Software Technology*, vol. 52, no. 10, pp. 1023–1043, Oct. 2010.
- [31] O. Levina and V. Stantchev, “Realizing Event-Driven SOA,” in *Fourth International Conference on Internet and Web Applications and Services (ICIW)*, 2009, pp. 37–42.
- [32] B. Michelson, “Event-driven architecture overview: Event-Driven SOA Is Just Part of the EDA Story.” Patricia Seybold Group, 02-Feb-2006.

- [33] D. Ravier, "A Service Oriented Framework Architecture for Intelligent Video Surveillance Systems," in *Fifth International Conference on Digital Telecommunications (ICDT)*, 2010, pp. 123–127.
- [34] K. Vidackovic and A. Weisbecker, "A Methodology for Dynamic Service Compositions Based on an Event-Driven Approach," in *2011 Annual SRII Global Conference*, 2011, pp. 484–494.
- [35] M. Wieland, D. Martin, O. Kopp, and F. Leymann, "SOEDA: A Method for Specification and Implementation of Applications on a Service-Oriented Event-Driven Architecture," in *Business Information Systems*, W. Abramowicz, Ed. Springer Berlin Heidelberg, 2009, pp. 193–204.
- [36] C. Zang, Y. Fan, and R. Liu, "Architecture, Implementation and Application of Complex Event Processing in Enterprise Information Systems Based on RFID," *Information Systems Frontiers*, vol. 10, no. 5, pp. 543–553, Nov. 2008.
- [37] MuleSoft, "Mule ESB," 2015. [Online]. Available: <http://www.mulesoft.org/>. [Accessed: 28-Feb-2015].
- [38] S. Ried, "The Forrester Wave: Hybrid 2 Integration, Q1 2014," Forrester Research, USA, Feb. 2014.
- [39] EsperTech, "Esper - Complex Event Processing," 2015. [Online]. Available: <http://www.espertech.com/esper/>. [Accessed: 28-Feb-2015].
- [40] Eclipse Foundation, "GMP: Graphical Modeling Project," 2015. [Online]. Available: <http://www.eclipse.org/modeling/gmp/>. [Accessed: 15-Feb-2015].
- [41] Eclipse Foundation, "Epsilon," 2015. [Online]. Available: <http://www.eclipse.org/epsilon/>. [Accessed: 28-Feb-2015].
- [42] J. Boubeta-Puig, G. Ortiz, and I. Medina-Bulo, "A Model-driven Approach for Facilitating User-friendly Design of Complex Event Patterns," *Expert Systems with Applications*, vol. 41, no. 2, pp. 445–456, Feb. 2014.
- [43] R. Gad, J. Boubeta-Puig, M. Kappes, and I. Medina-Bulo, "Hierarchical Events for Efficient Distributed Network Analysis and Surveillance," in *Proceedings of the 2nd International Workshop on Adaptive Services for the Future Internet and 6th International Workshop on Web APIs and Service Mashups*, Bertinoro, Italy, 2012, pp. 5–11.
- [44] S. Sen, L. Stojanovic, and I. Obradovic, "Interaction Pattern Editor," 10-Apr-2012. [Online]. Available: <http://alert-project.eu/sites/portal2-alert.atosorigin.es/files/content-files/download/Interaction%20Pattern%20Editor.pdf>. [Accessed: 17-Feb-2015].
- [45] MuleSoft, "Anypoint Studio," *Mule ESB Developer Community*, 2015. [Online]. Available: <http://www.mulesoft.org>. [Accessed: 28-Feb-2015].