A Middleware Platform to Federate Complex Event Processing

Fawaz Paraiso, Gabriel Hermosillo, Romain Rouvoy, Philippe Merle, Lionel Seinturier

A Middleware Platform to Federate Complex Event Processing

Fawaz Paraiso, Gabriel Hermosillo, Romain Rouvoy, Philippe Merle, Lionel Seinturier

University of Lille & Inria Lille - Nord Europe
LIFL UMR CNRS 8022, France
Email: firstname.lastname@inria.fr

***Abstract*—Distributed systems like crisis management are subject to the dissemination of a huge volume of heterogeneous events, ranging from low level network data to high level crisis management intelligence, depending on the role of the rescue teams involved. In such systems, *Complex Event Processing* (CEP) has emerged as a solution to detect and react (in real-time) to complex events, which are correlations of more primitive events. Although various CEP engines implement the support for dealing with the business heterogeneity of events, the technological integration of these events remains uncovered. Therefore, in this paper we introduce DiCEPE (*Distributed Complex Event Processing Engine*), a platform which focuses on the integration of CEP engines in distributed systems. DiCEPE provides a native support for various communication protocols in order to federate CEP engines and ease the deployment of complex systems-of-systems. We illustrate our proposal using a nuclear crisis management scenario and show how DiCEPE leverages the coordination and the federation of different CEP engines.**

***Keywords*-CEP; SOA; middleware; federation; component;**

I. Introduction

Nowadays, there is a huge amount of information sources everywhere around us, and heterogeneity is the rule among them. These events are used by distributed applications such as Air-Traffic Control Systems [16], Automated Banking Systems [9], Tracking Roaming Cellular Telephones [15], Retail Point-of-Sale Terminals [3] or Global Positioning Systems [23], that require them to be filtered and correlated for complex pattern detection and transformed to new events that reach a semantic level appropriate for higher-level applications.

The need for real-time processing of information is relevant for many systems, as not only the content but also the context in which it was created, determine its value. Having such contextual information in real-time may substantially increase the performance of an application, improving its liveliness since it can better react to the constantly changing situations. One example of such an application can be seen in a nuclear crisis management scenario, where there are several factors that may alter the outcome, and where a lot of information needs to be analyzed and processed (*e.g.*, weather forecasts and radiation surveys), while at the same time, several different teams (*e.g.*, policemen, firemen, medics) are cooperating in the same zone and depend on the decisions made using that information.

One of the problems in that kind of scenarios is the overwhelming amount of data that has to be considered at the same time, which may lead to information overload. The way in which data is handled is very important, and processing it using only one Complex Event Processing (CEP) engine may be a dangerous practice [8], as it may miss some important information due to being overwhelmed by all the events. Moreover, in a crisis management scenario we can see that there are several groups that are sharing the same information to achieve a common goal, however, the way in which each of them manages and interprets the information is different from the others. This could lead to incompatibilities among the groups.

In this paper we present DiCEPE, a platform that focuses on the integration of CEP engines in distributed systems, and which is capable of using various communication protocols in order to federate CEP engines and ease the deployment of complex systems-of-systems. DiCEPE uses the configuration capabilities provided by OASIS's Service Component Architecture (SCA) [11] to create a distributed environment of CEP engines that can prevent information overloading by sharing the load among different CEP engines. This approach also allows multiple domain-specific CEP engines to be included in the environment and to be managed directly by the domain experts, while at the same time sharing the global-level events and processing.

The reminder of this paper is structured as follows. In Section II we give an overview of some background concepts that we use in our proposal. Next, Section III presents the motivation of our work. In Section IV we introduce the DiCEPE platform. Then, in Section V we describe the integration of different CEP engines with DiCEPE. Next, in Section VI we present the validation of our approach. In Section VII we discuss some related work. Finally, we conclude our work in Section VIII.

II. Background

In this section we give a brief introduction to some of the concepts and technologies that we use throughout this paper, in order to facilitate the understanding of how they are used with the DiCEPE platform.

A. Complex Event Processing

Complex Event Processing (CEP) has been proposed as a new paradigm for real-time event-based applications [8]. It is designed to match event correlations, called *event patterns*, from a constant flow of events, called an *event stream*. One key aspect of CEP is the ability to define reactive rules corresponding to *event pattern* definition and triggering a process in real-time when an *event pattern* is detected. CEP is used in a wide diversity of applications, which have to deal with voluminous streams of incoming data, the complexity of processing and time response [17], [22], [25]. There are many domains in which CEP is actually used, like network management, traffic monitoring, or fraud detection, and it has drawn the attention of many research projects [2], [6] and commercial products [17], [22].

B. SOA and SCA

Service-Oriented Architecture (SOA) is a paradigm for the realization and maintenance of business processes that span large distributed systems [10]. It is independent of the development technology and comprises loosely coupled, highly inter-operable application services, which inter-operate based on a formal interface definition independent of the underlying platforms and programming languages. The DiCEPE platform uses a programming model based on the SOA paradigm, called Service Component Architecture (SCA).

SCA [11] is a set of specifications for building distributed applications and systems using SOA. It is neutral with respect to programming languages, and can be implemented in any language that supports it. SCA targets the heterogeneous composition of various interface definition languages (*e.g.*, WSDL, Java), implementation technologies (*e.g.*, Java, Spring, BPEL, JavaEE, C++, COBOL, C), and binding technologies (*e.g.*, Web Services, JMS).

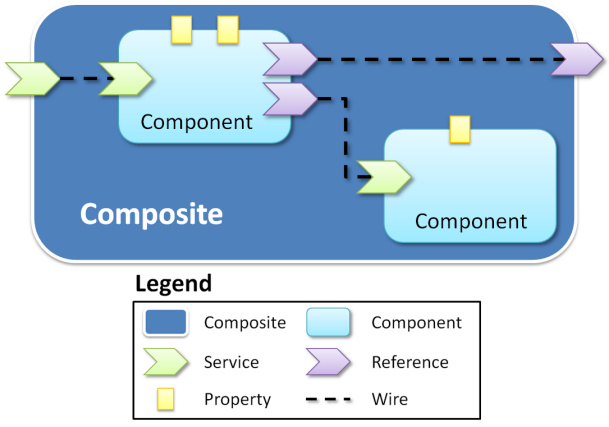

Figure 1. Overview of an SCA Diagram

As illustrated in Figure 1, the basic SCA building blocks are software components [24], which provide services, require references and expose properties. The references and services are connected by wires. SCA specifies a hierarchical component model, which means that components can be implemented either by primitive language entities or by subcomponents. In the latter case the components are called composites. Any provided services or required references contained within a composite can be exposed by the composite itself by means of promotion links. To support service-oriented interactions via different communication protocols, SCA provides the notion of binding. For SCA references, a binding describes the access mechanism used to invoke a remote service. In the case of services, a binding describes the access mechanism that clients use to invoke the service. DiCEPE is implemented on top of an SCA platform called FraSCAti, which we describe below.

C. FraSCAti

FraSCAti [20] is an open source platform for deploying and executing SCA-based applications. It introduces reflective capabilities to the SCA programming model, and allows dynamic introspection and reconfiguration via a specialization of the Fractal component model [1]. These features open new perspectives for bringing agility to SOA and for the run-time management of SCA applications. The platform itself is built as an SCA application (its different subsystems are implemented as SCA components). This provides an homogeneous view of a middleware software stack where the platform, the non-functional services, and the applications are uniformly designed and implemented with the same component-based and service-oriented paradigm. With FraSCAti, the structure of an SCA application can be discovered at run-time using introspection, and modified dynamically to add new services, or even reconfigured to take into account new operating conditions.

The component-based approach of the FraSCAti platform has the following characteristics:

- **Reusability**: Components can be re-used in different composites.
- **Substitutability**: Alternative implementations are easy to insert, specified interfaces are available, run-time component replacement mechanisms exist, and it has the ability to verify and validate substitutions.
- **Composability**: A subset of different components can be combined to create complex functional solutions that meet application requirements and system constraints, without the need to load all the existing components every time.
- **Extensibility**: The platform can be easily extended by adding components that provide additional functionality in a seamless way.

Finally, FraSCAti provides a unified way to build applications that communicate among each other in an heterogeneous way, as it supports a wide variety of communication protocols (*e.g.* Web Services, REST, JMS, JNA, UPnP).

III. Motivation

An emerging characteristic of event processing is its ubiquity: events are everywhere. An example of this can be found in a nuclear crisis management scenario. In this scenario, the multiplicity and diversity of the actors involved, as well as the volume and heterogeneity of information, the critical dependencies between actions, and the dynamics of the scenario create complex situations. These situations can be modeled as patterns that represent topics of interest which need to be detected nearly in real-time (*e.g.*, evacuation of the perimeter, distribution of iodine capsules, information to the public by the media). Events from heterogeneous sources (*e.g.*, firemen, policemen, army) can be combined to detect further situations of interest (*i.e.*, complex events).

For instance, let us consider a scenario where, after a nuclear crisis alert, the evacuation of the population goes smoothly. There are no victims, there is no change in the force or direction of the wind, no rain, nor any alert regarding high radioactive levels. Suddenly, the nuclear plant teams detect a radioactive leak, thanks to an alert given by a high pressure sensor. The throttle valve is open and is not responding to the remote commands to get it closed. The teams realize that there is a risk of radioactive gas leaking into the atmosphere, so they alert the manager of the nuclear plant. This use case is studied in the context of the ANR SocEDA project[1].

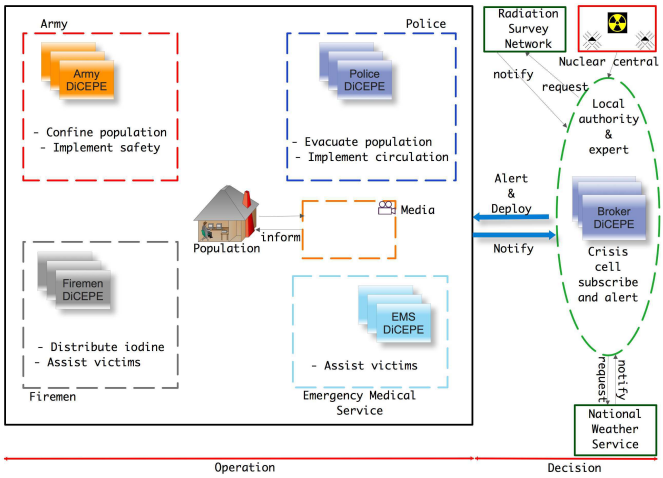

Figure 2. Nuclear Crisis Scenario

In this case, we are particularly interested in decision and operation activities that occur outside the plant, as illustrated in Figure 2. The manager of the nuclear plant informs the representative of the national authority, who activates the Emergency Intervention Plan [14]. The manager also informs the representative of the electric power company, and a crisis cell is formed, led by the local authority. The crisis cell alerts the field actors (*e.g.*, firemen, policemen, army, Emergency Medical Services) and calls for support from the Radiation Survey Network (RSN) and the National Weather Service (NWS) for measurements. The crisis cell also alerts the media and sets off the siren so that the population can learn that they have to stay indoors and listen to the media. The field actors are deployed and regular updates are given to the media, so that the population may remain informed.

There are several factors that may affect the way in which the crisis management is carried out, and so, in order to make the correct decisions during the execution of the Emergency Intervention Plan in this scenario, we need a continuous monitoring of the following information: radioactivity level, wind force, wind direction and precipitation. A change in the wind speed or direction would mean that the evacuation perimeter and routes would have to be readjusted. A change in the radioactivity levels may trigger additional security measures to be considered and more aggressive actions to be taken into account.

As for the actors of the scenario, we can separate them in two main groups according to their role: Management and Operation. The *Management* group is in charge of evaluating all the information surrounding the situation, analyzing the possible solutions and deciding the better way to proceed. It is formed by the local authority and the members of the crisis cell. These actors are the heads of the whole process and are only there to manage the situation. They do not interact directly with the crisis site, however, any situation that may require a change in the normal execution of the plant has to be analyzed and decided by them. On the other hand, the actors in the *Operation* group are the ones that interact directly with the crisis site, such as policemen, firemen, the army, and the Emergency Medical Services. They execute any orders received from the *Management* group. Even through they have a pre-established sequence of tasks to execute, their behavior can be modified by a *decision making* actor if the situation changes.

Each subgroup of actors (*e.g.*, the army or the police) will have their own CEP engine to manage the events related to their domain. A domain represents a complete run-time configuration, potentially distributed over multiple run-time nodes where statement rules were managed and processed. Each subgroup creates their own rules and complex events according to the goals of their domain. The problem is that when different actors are working together to achieve a common goal, we can no longer maintain a separate event management for each domain. In this case, we need to combine the events from all the actors involved in order to improve the performance of the collaboration, but at the same time, we need to keep the events that only concern a single domain from spamming the global event management, and leave the control of domain-specific rules and events to be managed separately, as is illustrated in Figure 3.

[1]SocEDA (SOCial Event Driven Architecture)
http://tinyurl.com/SocEDA

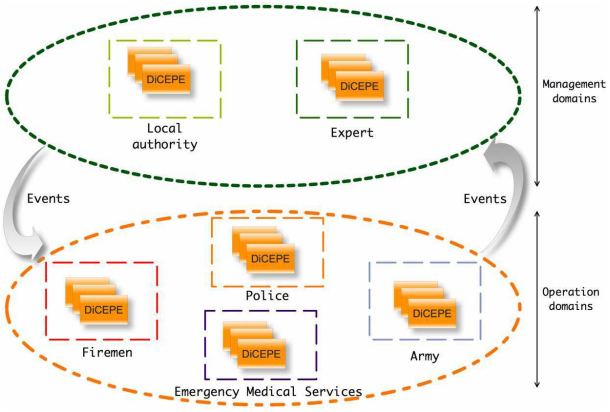

Figure 3. Communication between domains

Moreover, we also need to allow some sort of hierarchy, so that the *Management* group may receive the events that it needs for the decision making process, from the *Operation* groups and from external sources (*e.g.*, RSN, NWS), and be able to modify the global event management of the *Operation* groups according to their decisions, without interfering with domain-specific rules.

Given the presented scenario, we find that there are many challenges that need to be faced, in order to accomplish a successful federation of complex event processing systems.

- **Communication heterogeneity**: The nuclear crisis management puts together a large number of actors and CEP engines that need to interact through the network, according to the scenario previously described (see Figure 2). The communication among them is essential, and for different circumstances we may need different types of communication protocols. For instance, when interacting in a private network, an asynchronous protocol, such as JMS, may be much faster and as reliable as a synchronous one. However, when leaving the private network, it may be blocked due to firewall restrictions, and its reliability may suffer as well. In that case, a synchronous approach like REST could become useful.
- **Heterogeneous CEP**: The actors have different tasks which may generate multiple complex events. These events can then be processed by other CEP engines that may belong to the same or to different domains. Each group of actors can have different kinds of complex event processing engines. In this point, the challenge consists in supporting the interaction among heterogeneous complex event processing engines.
- **Scalability**: This distributed system will process a huge volume of events, ranging from low level network data to high level crisis management intelligence. Enabling the efficient detection of such situations in real-time and under critical conditions is crucial.
- **Adaptability**: In the scenario, the Emergency Intervention Plan can change during the crisis. Therefore, the possibility to deploy new rules at run-time is essential for the correct federation of complex event processing systems. The domain-specific and engine-constraint rules will ensure the processing of new conditions in the network nodes.

This paper brings forward a solution to these challenges, called DiCEPE (*Distributed Complex Event Processing Engine*). With DiCEPE, we allow the federation of distributed CEP engines that can deal with the heterogeneity of the event's sources and communication protocols, address scalability, provide the means to hierarchically control, and adapt the rules of the different CEP engines in the distributed environment. This approach will allow a domain-specific engine to be managed by the experts of the domain, while providing at the same time a way to create collaborations among different domains.

IV. The DiCEPE Platform

In this section we present the DiCEPE platform. We begin by giving an overview of the solution and then we present the platform's architecture. Finally, in the following section we will describe how DiCEPE can integrate different CEP engines, thanks to its architecture.

A. DiCEPE Overview

DiCEPE is based on the concepts introduced by the Event Processing Network (EPN) [8], however it evolves those concepts by incorporating the advantages of SCA. DiCEPE inherits the flexibility and adaptation facilities provided by FraSCAti, which allow it to provide a complete solution to federate complex event processing systems.

To understand the concepts that rely under the construction of DiCEPE, we begin by explaining how an EPN works. An EPN is a conceptual model, that describes a set of Event Processing Agents (EPA), Event Producers and Event Consumers all connected by a set of Event Channels (EC) [5]. The event producers are entities that create events, while the event consumers are the entities that receive such events. The EPA behaves as both, consuming events generated by producers, and then forwarding them or creating new events to other consumers. The goal of the EPA is to filter, match and derivate events, according to the business rules. A complex event processing engine is an example of an EPA. Basically, the EPN describes how events received from producers are processed by the agents, who perform transformations, validations or enrichment on them, and finally directed to the consumers. An example of such a network is presented in Figure 4.

The EPN is an abstract model, since it abstracts the features of the input, processing and output elements of an event processing system. Guided by these concepts, our DiCEPE architecture identifies the abstract elements or components and then provides a concrete and flexible way to

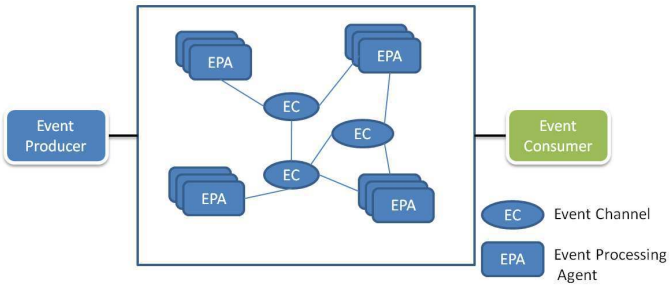

Figure 4. Event Processing Network

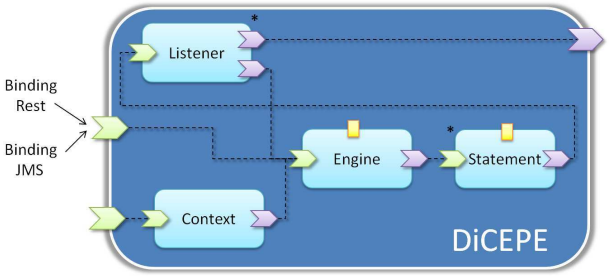

Figure 6. Overview of the DiCEPE Architecture

federate complex event processing engines. In our approach, all the components of the EPN are transformed into DiCEPE components, the different instances of EPA are wrapped by a DiCEPE composite and the communication among them is managed using SCA bindings, as is presented in Figure 5. This approach allows us to overcome some of the weak points of the EPN model, such as heterogeneity and adaptability, as we will present in the following sections.

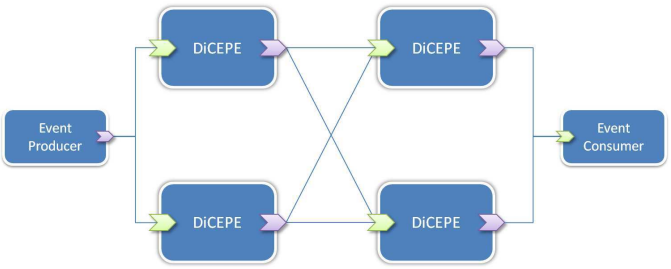

Figure 5. Distributed Complex Event Processing Engine

B. DICEPE Architecture

DiCEPE is a platform for federating CEP engines, with synchronous and asynchronous communication protocols. Its architecture is based on the EPN, but it improves it by filling the gaps that are still present in the EPN model. Since our solution deals with many CEP engines, we describe it in terms of a concrete SOA platform. The proposed solution is implemented on top of FraSCAti [20]. However, most of the concepts used in DiCEPE are generic, and we believe a similar implementation can be done on top of other component frameworks as well. The architecture of DiCEPE is composed of four parts: (i) Engine, (ii) Statement, (iii) Listener, and (iv) Context; as presented in Figure 6.

- **Engine**: This component acts as the engine instance, by which Statement components, events, and outputs (Listener component) are registered.
- **Statement**: A Statement component is used for quering the inbound event streams. This component is registered within the Engine component. The Engine component is connected to one or many Statement components.
- **Listener**: A Listener component generates a new complex event when an action is detected. Each Listener component is associated to a Statement component.
- **Context**: A Context component collects information, like the number of statement rules deployed in the engine at run-time.

In addition, DiCEPE supports both synchronous and asynchronous communication. This is useful, for example in the crisis scenario, where in a private network the JMS protocol is sometimes more appropriate for communication among CEP engines given its speed and reliability [4]. But, when we scale the scenario to a city using the Internet, the REST protocol is the better suited, since it is less likely to get blocked by a Firewall (since it uses HTTP), and also has great reliability. To retrieve information from the NWS and the RSN, we can use Web Services. There is no universal communication protocol that takes into consideration all those aspects. However, the DiCEPE platform provides access to several communication protocols which can be used as needed.

To summarize, by allowing the use of both communication protocols, synchronous and asynchronous, DiCEPE provides a solution to the **Communication heterogeneity** challenge identified in Section III.

The deployment of statement rules at run-time is very important for the crisis management plan, because it can change depending on the circumstances. The DiCEPE platform inherits its reconfiguration capabilities from FraSCAti [20], so a component can be stopped, modified and restarted at run-time, without service downtimes. Therefore a domain-specific rule can be changed and deployed at run-time as well, allowing it to be adapted to the new context. With this capability, DiCEPE addresses the **Adaptability** challenge presented in Section III.

V. Integration

This section describes how the integration of different CEP engines with the DiCEPE platform is achieved. As described in Section II, FraSCAti provides a component-based programming model which simplifies the development, assembly, deployment and management of composite applications. The DiCEPE platform facilitates the integration

of complex event processing engines. Moreover, the DiCEPE platform also encapsulates the event processing technology, so that each node in the network is running a DiCEPE runtime environment. In the following subsection we describe the integration of DiCEPE with two open source CEP engines (Esper, Etalis).

A. Integration with the Esper CEP Engine

Esper[2] is an open source engine that combines Event Stream Processing (ESP) and CEP capabilities. It is available as Java source and C# .Net source (NEsper). Their Event Processing Language (EPL) is used to express filtering, aggregation and joins, over multiple event streams, while the pattern language is used to define more complex patterns on different types of events. Esper also includes a historical data access layer to connect to the most popular databases, making the possible to combine historical data with real time data in one single query.

To describe how the Esper engine is integrated into DiCEPE, it's important to understand Esper's architecture, which is shown in Figure 7.

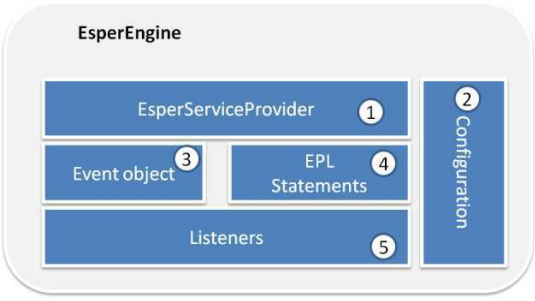

Figure 7. Overview of Esper Engine Architecture

In the figure we can see the different elements of the Esper engine:

1) **The EPServiceProvider**, which acts as the Engine.
2) **The Configuration**, which sets the engine configurations.
3) **The Event object**, which is an object that represents an event.
4) **The EPLStatement**, which are queries written in EPL.
5) **The UpdateListener**, which receives updated data as soon as it is processed by the statement.

These elements are integrated into DiCEPE in the following way: The Engine composite encapsulates both the EsperServiceProvider (**1**) and the Configuration components (**2**), having the EventObject as property (**3**). The Statement component is associated to the EPLStatement (**4**). Finally, the Listener component represents the UpdateListener (**5**). The Engine component exposes the engine services with different communication protocols (*e.g.*, REST, WS-Notification, JMS).

All the relationships in an SCA composite are expressed using the SCA Assembly Model [12]. Here is a slightly simplified example of how this file might look for the integration of the Esper CEP engine shown above:

```
1  <composite xmlns="http://www.osoa.org/xmlns/sca/1.0"
2    xmlns:frascati="http://frascati.ow2.org/xmlns/sca/1.1"
3    xmlns:stmt="statements" name="fireStation">
4
5    <component name="esper-engine-fireStation"
6             constrainingType="esper:Engine">
7      <implementation.java
8          class="org.ow2.frascati.esper.
9                impl.EsperEngineImpl.java"/>
10     <property name="events" type="esperEvents:Events">
11       <esperEvents:Events>
12         <esperEvents:Event
13           esperEvents:event-type='CLASS'>
14         org.ow2.frascati.soceda.eventBean.AgentLocation
15         </esperEvents:Event>
16       </esperEvents:Events>
17     </property>
18     <service name="DiCEPEFireStation">
19       <interface.java
20         interface="org.ow2.frascati.esper.api.
21                    EsperEngine"/>
22       <frascati:binding.rest
23         uri="http://dicepe-firestation.soceda.cloudbees.
24                     net/EsperEngine"/>
25     </service>
26   </component>
27
28   <!--Component statement-->
29   <component name="fireStation-stmt">
30     <implementation.composite
31       name="stmt:fireStationStmt"/>
32     <reference name="engine" autowire="true">
33       <interface.java
34         interface="org.ow2.frascati.esper.
35                    api.EsperEngine"/>
36     </reference>
37
38   <!--Simulation Rest Services -->
39     <reference name="siafu">
40       <interface.java
41         interface="org.ow2.frascati.broker.service.
42                    Simulation"/>
43       <frascati:binding.rest uri="/simulation"/>
44     </reference>
45   </component>
46 </composite>
```

Listing 1. Descriptor for Firemen Composite with Esper.

Like all SCA Assembly Model configurations, this one wraps its contents in an SCA composite (*cf.* lines 5-46). In the example shown here, a component element describes each of the two components in this composite. The first component is implemented in Java as indicated in lines 7-9. The second is embedded as a component as indicated in lines 30-31. Even though each component has it's services (*cf.* lines 18-25) and references (*cf.* lines 32-44) shown in the diagram, some are explicitly specified in these component elements, while others are not. Instead, the FraSCAti runtime can discover them and then choose the appropriate wires.

After both components have been defined, the service provided by the composite itself is specified using the service element (*cf.* line 18).

[2]http://esper.codehaus.org

B. Integration with the Etalis CEP Engine

Etalis[3] is an open source system for Complex Event Processing which includes two languages: the Etalis Language for Events (ELE) and the Event Processing SPARQL (EP-SPARQL). The Etalis CEP engine is implemented in Prolog, and supports reasoning about events, contexts, and real-time complex situations (*i.e.,* Knowledge-based Event Processing).

The DiCEPE platform provides a convenient way to access native libraries with pure Java code, using the binding of JNA in FraSCAti with the Prolog environment. In our case, we built an adapter component for the Etalis Engine, which provides a convenient way for getting events in and out of the Etalis Prolog core, as shown in Figure 8. The Engine component embeds the PrologEngineWrapper, while the Listener component is associated to the EtalisEventListener.

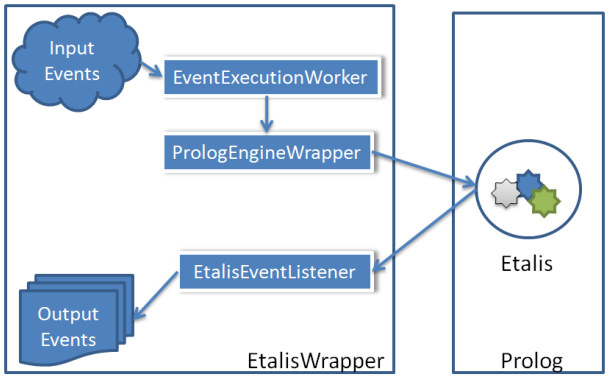

Figure 8. Overview of the Etalis adapter

With this, we have shown that the DiCEPE platform can integrate two CEP engines, written in different languages (Java for the Esper engine and Prolog for the Etalis engine). This addresses the **Heterogeneous CEP** challenge presented in Section III.

VI. Validation

This section describes our simulation results, which study the heterogeneity of events and their interactions via different communication protocols, as well as the performance and scalability of DiCEPE. Given the complexity of crisis management and to facilitate the understanding, we focus on one single scenario.

A. Context

This use case consists in the construction of a simulation tool to validate the DiCEPE platform. The simulation tool was created based on *Siafu*[4], an open source context simulator written in Java, which allows to integrate real and simulated contexts to enrich the testing capabilities. To obtain a simulated context, we need to follow three steps. First, the behavior of individual agents is modeled in a separate agent model. Then, the environment is handled in a world model, including possible random events. Finally, we get a context model which is used for defining how context data is simulated. In our simulation tool we designed a small city, where we have police stations, fire stations and an army camp. There are also two nuclear centrals nearby which have several sensors attached to them. Finally, the National Weather Service (NWS) and the Radiation Survey Network (RSN) are also reachable for information. The case study shown in Section III (*cf.* Figure 2) illustrates this scenario.

To evaluate our solution, we try to find an answer to the following questions:
(i) Does DiCEPE introduce much overhead?
(ii) What is the scalability of the system?
To focus on the real performance of the DiCEPE platform, without the overhead of Internet lag, all the benchmark experiments were performed on a local machine, using an HP Z4000 Workstation with a 2.67 GHz Intel(R) Xeon processor, 16 GB RAM, Ubuntu Server 3.0.0-12 64 bit and Oracle Java 1.6. In addition, another version of this implementation was also deployed to a cloud environment, as described in the following subsection.

B. DiCEPE deployment

Our solution was deployed on a full cloud environment, that is publicly accessible on the Web[5]. For this, we used a public Platform as a Service (PaaS) provider, called *CloudBees*[6], which is basically a Virtual Machine running a Linux distribution, with a Java virtual machine and a Web application container, all hosted in the cloud and offered as a service. In this case, the PaaS hardware resources (*e.g.,* RAM, CPU, Storage) are provided by an external Infrastructure as a Service (IaaS) provider, independent from the PaaS. In Table I we describe the characteristics of the PaaS that we used to deploy the different instances of the DiCEPE platform that are required to run our scenario[7].

Table I
THE CLOUD PLATFORM USED

Cloud		**PaaS software stack**	
Provider	Location	JRE	Web container
CloudBees	Amazon EC2 US West	JVM 6	Apache Tomcat 6.0.32

C. DiCEPE Cost Analysis

To evaluate the overhead of the DiCEPE platform, 1,001,946 events were collected from our simulation tools, which represents about 1GB of data to be processed by the CEP engines. We then fed these events to DiCEPE in

[3]http://code.google.com/p/etalis
[4]http://siafusimulator.sourceforge.net
[5]http://dicepe-broker.soceda.cloudbees.net
[6]http://www.cloudbees.com
[7]The IaaS hardware information (CPU, RAM, Storage and OS) are not available to end users.

order to determine which houses were not covered by the firemen. We evaluated two implementations of this scenario where the events were processed: *i)* natively with the *Esper*[8] CEP engine, and *ii)* with DiCEPE integrating Esper and implemented on top of FRASCATI 1.5.

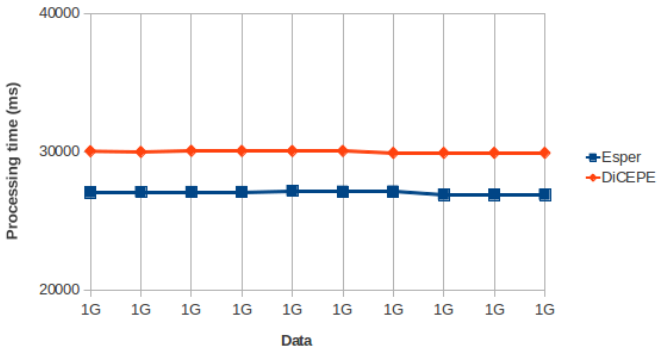

Figure 9. Cost analysis

The scenario was executed ten times on each of the two implementations. As shown in Figure 9, the execution time of each request is stable for both implementations, which means that they are both deterministic.

In Table II, we present the results of the average execution time for each implementation, as well as the mean overhead introduced by the SCA run-time used by DiCEPE.

Table II
EXECUTION TIME AND OVERHEAD

Implementation	Avg. exec. time	SCA overhead
Esper	27 sec	-
DiCEPE (Esper + FraSCAti)	30 sec	11%

This benchmark shows that there is an overhead introduced by adding the FraSCAti layer, the execution time is still acceptable and the benefits provided by the platform (*cf.* Section II) outweigh the difference in the execution time. Even if FraSCAti could still be optimized in order to reduce its overhead, we can already consider that DiCEPE is ready for intensive real-time distributed complex event processing applications.

D. DiCEPE Scalability

Our second evaluation of DiCEPE was lead towards scalability. For this, we used the scenario of our case study (*cf.* Section III), and we wrote a benchmark that mines firemen events. We simulated several firemen inside one big Fire Station using our simulation tool. Each fireman had devices which received notifications and sent events to their DiCEPE platform, which embedded one CEP engine (for this benchmark the CEP engine used was Esper). The firemen's DiCEPE received events, processed them and generated complex events, which were sent to the DiCEPE broker. This is done for the rest of the actors as well, as shown in Figure 10. The goal of the broker is to process the events of the specialized DiCEPE entities to find information which is relevant to the whole situation. Also, thanks to the capabilities provided by DiCEPE to deal with heterogeneous communication, all the entities can interact using different types of communication.

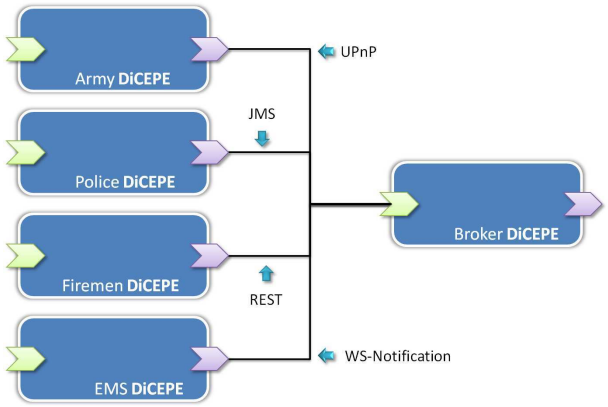

Figure 10. Distributed CEP engine in a crisis management

To evaluate this scenario, we used two different datasets of event generators: one with 10,000 (see Figure 11) and a second one with 15,000 (see Figure 12), which generated around 500,000 and 750,000 events respectively (50 events per generator, without considering the start and finish notifications). As can be seen in Table III, the processing time for each event remained stable and very low during both benchmarks (around a 10th of a millisecond), despite the fact that the average number of simultaneous sessions had a significant increase of about 50% (from 89 with the first dataset, to 135 with the second). It should also be noted that there were no failures nor events lost during the executions of these tests.

Table III
BENCHMARK RESULTS

Generators	Events	Failures	Avg. sessions	Avg. response
10,000	500,000	0	89	0.113 ms
15,000	750,000	0	135	0.142 ms

Overall, these benchmarks show that the response time is negligible despite the scalability, knowing that in real life we could not have more than 10,000 firemen inside one single Fire Station. For this case study we did not use any cache, which means that the response time can also be improved. With this data, we may conclude that the DiCEPE platform scales well, which addresses the **Scalability** challenge presented in Section III.

[8]http://esper.codehaus.org

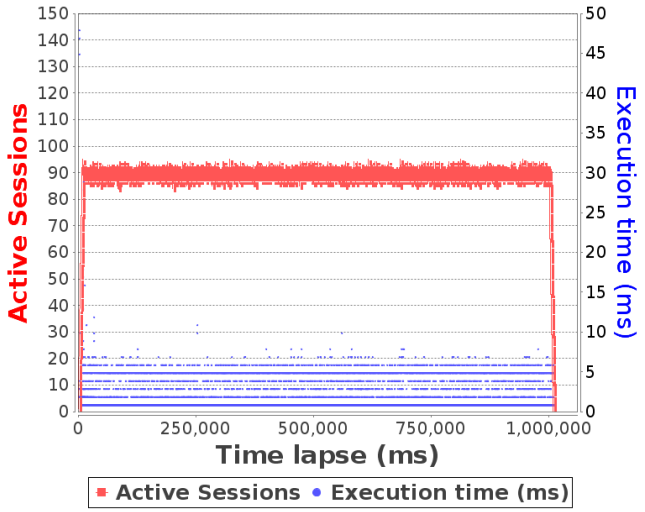

Figure 11. Response time during simulation with 10,000 firemen

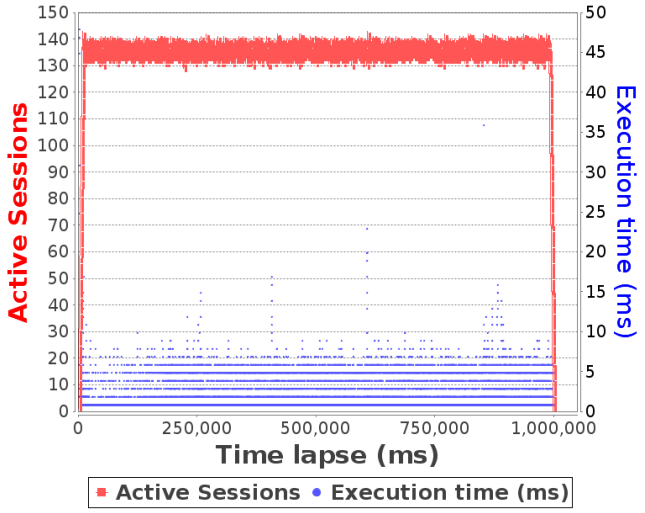

Figure 12. Response time during simulation with 15,000 firemen

VII. Related Work

In this section we will present some of the related work from different fields of research that are relevant to our solution.

We already introduced the Event Processing Network (EPN) in Section IV, and explained how the DiCEPE architecture is based on its four components: the Event Producer (EP), the Event Consumer, the Event Processing Agent (EPA), and the Event Channel [13], [21]. However, the EPN model is abstract and only contemplates the independent interaction of isolated EPAs, and does not take into account a distributed architecture.

There are already some works that start dealing with Distributed CEP. For instance, in [19] the authors present a distributed approach for distributed event detection, however they do not deal with CEP heterogeneity, and they do not provide the dynamic rule adaptation provided by DiCEPE. Another approach was proposed by [18], which takes into account the heterogeneity of event processing engines in their distribution techniques. However, their solution does not address scalability and does not provide a real way to interconnect heterogeneous components within a common platform.

CEP systems in general, both open research and commercial, propose an event channel to connect the event consumers, providers and processing agents [7]. However, none of them provide synchronous and asynchronous communication protocols, and as we previously presented in this paper, distributed systems perform better when both protocols are available. Communication in distributed systems is always based on low-level message passing, as offered by the underlying network. DiCEPE makes it easier to deal with the numerous levels of communication as well as with all the issues involved since it supports both, synchronous and asynchronous communication protocols (*e.g.*, JMS, SOAP, REST).

VIII. Conclusion

In this article we presented DiCEPE, a platform that offers interoperability for Distributed Complex Event Processing engines, via federation. This platform focuses on providing a very flexible component architecture, which supports the interaction of different complex event processing engines simultaneously, while enabling communication among them with a distributed system and deployment. Thanks to its distributed nature, DiCEPE also offers real scalability, and the evaluation results show that, despite the additional overhead generated by SCA, the execution times are still acceptable.

As of future work, we plan to continue our research in the following directions. First, since the Event Query Language (EQL) is not the same for all CEP engines solutions, we will integrate a Domain Specific Language (DSL) to express statement rules, with which we will be able to translate the DSL to the CEP specific language. Second, in the case of a geographically distributed CEP, the cloud is a good candidate. However, there are still some vendor lock-in problems with the existing solutions. This means that the deployment of the DiCEPE platform on heterogeneous cloud environments, either PaaS (Platform as a Service) or IaaS (Infrastructure as a Service), is still a challenge. As a third direction, we plan to add error handling capabilities to the DiCEPE platform to deal with distributed environments. Finally, we plan to design and evaluate a variety of distributed systems in heterogeneous cloud environments with large scale scenarios.

Acknowledgment

This work is partially funded by the French Ministry of Higher Education and Research, Nord-Pas de Calais

Regional Council and FEDER through the Contrat de Projets Etat Region Campus Intelligence Ambiante (CPER-CIA) 2007-2013, and the ANR (French National Research Agency) ARPEGE SocEDA project.

REFERENCES

[1] E. Bruneton, T. Coupaye, M. Leclercq, V. Quéma, and J.-B. Stefani. The FRACTAL component model and its support in Java: Experiences with Auto-adaptive and Reconfigurable Systems. *Softw. Pract. Exper.*, 36(11-12):1257–1284, Sept. 2006.

[2] S. Chakravarthy and Q. Jiang. *Stream data processing: a quality of service perspective : modeling, scheduling, load shedding, and complex event processing*. Advances in Database Systems. Springer, 2009.

[3] Christopher M. Benson. Clustering of retail terminals. Website http://www.google.com/patents/US20030110082, 2001.

[4] R. Eggen and S. Sunku. Efficiency of Soap Versus JMS. In *Proceedings of the International Conference on Internet Computing, IC '03*, pages 99–105, 2003.

[5] O. Etzion and P. Niblett. *Event Processing in Action*. Manning Publications Co., 2010.

[6] M. K. Kasi and A. M. Hinze. Cost analysis for complex in-network event processing in heterogeneous wireless sensor networks. In *Proceedings of the 5th ACM international conference on Distributed event-based system*, DEBS '11, pages 385–386, New York, NY, USA, 2011. ACM.

[7] D. Luckham and R. Schulte. Event Processing Glossary - Version 1.1. *Processing*, 1.1(July):1–19, 2008.

[8] D. C. Luckham. *The Power of Events: An Introduction to Complex Event Processing in Distributed Enterprise Systems*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2001.

[9] Mark S. Covert et al. Automated banking machine and system. Website http://acomplete.com, 2001.

[10] OASIS. Reference Model for Service Oriented Architecture 1.0. Website http://www.oasis-open.org/committees/download.php/19679/soa-rm-cs.pdf, August 2006.

[11] OASIS. Organization for the Advancement of Structured Information Standards.Web Services Business Process Execution Language Version 2.0 Standard. (OASIS) . Website http://www.oasis-opencsa.org/sca, 2007.

[12] OASIS. *SCA Service Component Architecture - Assembly Model Specification*, May 2011. Version 1.1.

[13] E. Rabinovich, O. Etzion, S. Ruah, and S. Archushin. Analyzing the behavior of event processing applications. In *Proceedings of the Fourth ACM International Conference on Distributed Event-Based Systems*, DEBS '10, pages 223–234, New York, NY, USA, 2010. ACM.

[14] S-Cube: Crisis Management Case Study. Website http://tinyurl.com/NuclearCrisis.

[15] R.-G. Reuven Della-Torre. Tracking Roaming Cellular Telephones Calls For Anti-Fraud and Other Purposes. Website http://www.freepatentsonline.com/y2007/0072587.html, 2007.

[16] Richard H. R. Harper et al. Document processing and data distribution system for an air traffic control Environment. Website http://bks9.books.google.as/patents/US5764508, 1998.

[17] RTView platform. SL RTView platform. Website http://www.sl.com/solutions/cep.shtml.

[18] B. Schilling, B. Koldehofe, U. Pletat, and K. Rothermel. Distributed heterogeneous event processing: enhancing scalability and interoperability of CEP in an industrial context. In *Proceedings of the Fourth ACM International Conference on Distributed Event-Based Systems*, DEBS '10, pages 150–159, New York, NY, USA, 2010. ACM.

[19] Schultz-Møller, Nicholas Poul and Migliavacca, Matteo and Pietzuch, Peter. Distributed complex event processing with query rewriting. In *Proceedings of the Third ACM International Conference on Distributed Event-Based Systems*, DEBS '09, pages 4:1–4:12, New York, NY, USA, 2009. ACM.

[20] L. Seinturier, P. Merle, R. Rouvoy, D. Romero, V. Schiavoni, and J.-B. Stefani. A Component-Based Middleware Platform for Reconfigurable Service-Oriented Architectures. *Software: Practice and Experience (SPE)*, 42(5):559–583, May 2012.

[21] G. Sharon and O. Etzion. Event-processing network model and implementation. *IBM Syst. J.*, 47(2):321–334, Apr. 2008.

[22] SpatialRules. SpatialRules. Website http://www.objectfx.com/geospatial-solutions-products/spatialrules.

[23] J. Spencer. *Global Positioning System: a field guide for the social sciences*. Blackwell Pub., 2003.

[24] C. Szyperski. *Component Software: Beyond Object-Oriented Programming*. ACM Press and Addison-Wesley, New York, NY, 1998.

[25] WSO2. WSO2 Complex Event Processing Server. Website http://wso2.com/products/complex-event-processing-server.