Enhancing Event Processing Networks with Semantics to Enable Self-Managed SEE Federations

Srdjan Komazec
Semantic Technology Institute
University of Innsbruck
Technikerstraße 21a
6020 Innsbruck, Austria
srdjan.komazec@sti2.at

Davide Cerri
Semantic Technology Institute
University of Innsbruck
Technikerstraße 21a
6020 Innsbruck, Austria
davide.cerri@sti2.at

ABSTRACT

As a common platform to easily exchange information, the Web is already overwhelmed with events which need to be processed in timely fashion. The opportunity to detect event occurrences in real time at a Web scale while correlating events from different sources has the potential to enable powerful applications. Recent contributions in the area of event processing provide a generic framework, usually referred to as Event Processing Networks. This paper proposes a grounding of the Event Processing Network conceptual model on Semantic Web technologies, in order to realize Semantically-enhanced Event Processing Networks (SEPNs). As a use case, the paper describes the integration of SEPNs with a Semantic Execution Environment (SEE), in order to achieve a suitable level of monitoring and self-management. While relying on the proposed monitoring ontology, we show how SEPNs are capable of observing events at the level of both single and federated SEE deployments and of responding accordingly.

Keywords

Event Processing, Semantic Web, RDF(S), SPARQL, Monitoring, Web Services, Semantic Execution Environment

1. INTRODUCTION

Events are becoming constituent citizens of the Web. Seen as a common platform to easily and cheaply exchange information, the Web is already overwhelmed with events such as dissemination of new multimedia content, readings from the Sensor Web, and social networking activity notifications. Those events can be observed through various channels, such as e-mails, RSS feeds and RESTful Web services, and are primarily targeted to human consumption. The impact of a potential solution to tame such *"Web of Events"* can be substantial, since there are already many events on the Web but too little meaning of them. The opportunity to detect event occurrences in real-time at the Internet scale while correlating events from different sources emitted in heterogeneous formats could open the doors for powerful applications.

Event processing is an established computing paradigm which provides approaches and techniques to process event streams and to provide responses in a timely fashion. In particular, recent contributions provide a generic framework for event processing, usually referred to as *Event Processing Networks*. The combination of event processing techniques with the events distributed across the Web comes as a natural fit. Event processing on the Web needs however to cope with the openness and heterogeneity which are typical of Web environment, and tackle issues related to identification and discovery of event sources, and heterogeneity reconciliation. Semantic Web technologies may help to overcome those problems by precisely defining notions in the event processing domain and using machine processable descriptions to resolve heterogeneities and automate event processing tasks. The usage of Semantic Web technologies comes even more natural if we consider that events annotated with structured metadata are starting to appear on the Web: for example, Twitter is planning to add structured metadata to annotate tweets[1], while the field of Semantic Sensor Web [16] represents an attempt to collect and process avalanches of data about the world on top of Semantic Web technologies.

In this paper we propose a grounding of the Event Processing Network conceptual model on Semantic Web technologies, in order to realize *Semantically-enhanced Event Processing Networks* (SEPNs). Events processed by SEPNs are defined as ontological instances, and the behavior of SEPN building blocks is defined in terms of appropriate Semantic Web query and rule languages. As a use case, we present the application of our SEPN approach to the domain of Semantic Web Services, and in particular to monitoring a federation of Semantic Execution Environments (SEEs). We therefore propose an architecture for integrating SEE instances and SEE federations with SEPNs, to achieve an appropriate level of reactivity and to enable adaptation and self-management functionalities.

The rest of this paper is structured as follows. In section 2 we introduce the concept of Semantically-enhanced Event Processing Networks and we present a language for specifying the behavior of event processing agents. In section 3 we describe the application of SEPNs to monitoring SEE federations, introducing an ontology that covers the domain and presenting an extended SEE architecture which

[1]`http://dev.twitter.com/pages/annotations_overview`

integrates SEPNs in order to achieve monitoring and reactivity capabilities. Furthermore, we describe an example of a possible scenario of using SEPNs in a federated SEE environment. Finally, in section 4 we give an overview of the related work, while in section 5 we outline future work directions and conclude the paper.

2. ENHANCING EVENT PROCESSING NETWORKS WITH SEMANTICS

The concept of EPNs has been introduced in the seminal work of Luckham [10] and further refined by Etzion and Niblett [7]. An EPN represents a flexible mechanism to describe and build data-flow networks composed of the following constructs: event producers and consumers, event channels, and event processing agents (EPAs). Depending on their functionality, the agents can be further classified into filtering agents, pattern detection agents and transformation agents (which in turn can be further classified into aggregation, splitting, composing and translation agents). Regardless of the specific type, agents are operating on top of events floating through the network by applying filtering, matching and derivation functions in either stateful or stateless mode.

An example of an arbitrary EPN is shown in Figure 1. The network is composed of three event processing agents performing composition, filtering and enrichment functions (where filtering and enrichment agents are connected in a sequence). The enrichment agent is operating on top of the global state used to inject new data in processed events. Events produced by `Event Producer 2` and `Event Producer 3` are transferred into the network through the `Channel`, which is used for routing purposes.

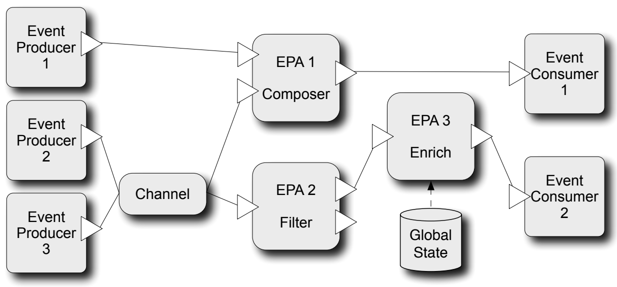

Figure 1: An Event Processing Network

Figure 2 depicts the architecture of an EPA. In a nutshell, an EPA is build around the concept of a data flow network in which `Input` and `Output Terminals` represent entering and exiting points for the processed events. The terminals are used to connect an EPA instance to the other EPAs and channels inside of an EPN. Inside of the EPA three building blocks connected in a cascade can be identified, each one associated with a specific function. The `Filtering` block is capable of applying filter conditions over the single event instances. The `Matching` block operates over a larger set of possibly diverse events while testing a denoted matching criterion over them and outputting a set of matching events satisfying the criterion. The `Deriving` block usually evaluates over the matching event set in order to derive a set of new events, while potentially taking into account data from

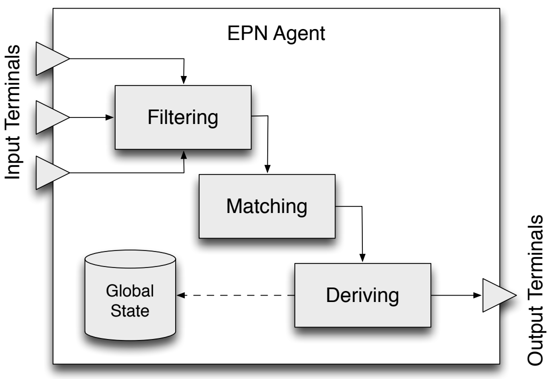

Figure 2: Architecture of Event Processing Agent

the `global state`. More information about the peculiarities of the building blocks and how specific EPA types rely on them is provided in [7].

Our Semantically-enhanced Event Processing Networks represent a specialization of EPN concepts built on top of Semantic Web technologies. Two main features distinguish SEPNs from other forms of EPNs: events processed by a SEPN are defined in the form of instances compliant with an ontology (see section 3.1), and the behavior of SEPN agents and event channels is described in terms of appropriate Semantic Web query and rule languages. In contrast to other possible EPN implementations, SEPNs provide powerful data processing mechanisms relying on the formal and established data models promoted by the Semantic Web, which in turn reduces technical barriers to applying such an EPN solution on the Web.

The SEPN approach that we propose relies on the established Semantic Web technologies. Events processed by SEPN network are straightforwardly defined in terms of RDF [11] graphs. The inner structure of the EPA (as presented in Figure 2) and the behavior of its building blocks reveal that SPARQL [15] can cover most of the requirements:

- *Filtering function.* The function can be treated as an `ASK` form of SPARQL query. Depending on evaluation outcome the event is routed to one of the output terminals, which must be additionally denoted.
- *Matching function.* The set of events treated by this function can be represented as an RDF graph (the union of RDF graphs denoting single events). The function tests a matching criterion over the graph and creates a matching set of events (a subgraph) which satisfies the criterion. A SPARQL query may be used to denote the conditions over the graph.
- *Derivation function.* The RDF subgraph from the previous step, together with the more stable knowledge from the global state represented by an RDF repository, is used to construct a new RDF graph. SPARQL enables the creation of new triples by relying on the `CONSTRUCT` form of a SPARQL query.

From the analysis above it can be concluded that large part of EPA functionality can be addressed with SPARQL query answering. However, in our case SPARQL query evaluation must efficiently accommodate fast changing data while retaining performance to fulfill (near) real-time reactivity (*continuous semantics*), in contrast to the traditional approach which assumes query evaluation over a stable data repository (*one-time semantics*). Adaptation of SPARQL to support *continuous semantics* and development of suitable query evaluation environments represents an ongoing work in the research community (e.g., C-SPARQL [2]). Furthermore, some of the functions (such as *derivation function*) may depend on a global state preserved outside of the primary agent data flow, which means that access to an external repository may be required.

Since the various EPN agent types require particular configurations of the aforementioned building blocks, a mechanism capable of enacting functional blocks is needed. In the next section we define a declarative language used to specify SEPN agent behavior.

2.1 SEPN Agent Functionality Specification Language

Each agent in a SEPN is configured with an expression following the formal grammar given in Listing 1, defined in EBNF[2]. The language enables the definition of the agent inner functions which heavily rely on SPARQL query language constructs. In accordance to the description above, the specification assumes implicit cascade event flow inside of an agent functionality specification, which relaxes constraints over respective input and output terminal declarations.

The definitions of `PrefixDecl`, `WhereClause` and `ConstructTemplate` are provided in [15]. An example usage of the SEPN Agent configuration expressions is given in section 3.3.

```
/* SEPN Agent functionality declaration */
SEPADecl     ::= PrefixDecl*
                 InTerDecl
                 OutTerDecl
                 FilterDecl?
                 MatcherDecl?
                 DeriverDecl?

/* Input and output terminals declaration */
InTerDecl    ::= 'INPUT TERMINAL'
                 (TermId EventTypeId)+
OutTerDecl   ::= 'OUTPUT TERMINAL'
                 (TermId EventTypeId)+

/* Filter function declaration */
FilterDecl   ::= 'FILTER' InputTerm* OutputTerm?
                 FilterExpr+
FilterExpr   ::= WhereClause

/* Matcher function declaration */
MatcherDecl  ::= 'MATCHER' InputTerm* OutputTerm?
                 MatcherExpr
MatcherExpr  ::= WhereClause

/* Deriver function declaration */
DeriverDecl  ::= 'DERIVER' InputTerm* OutputTerm
                 GlStateId? DeriverExpr
GlStateId    ::= 'GLOBAL STATE' IRIREF
DeriverExpr  ::= 'DERIVE' ConstructTemplate
                 'FROM' WhereClause
```

```
/* Helper expressions */
InputTerm     ::= 'INPUT' TermId
OutputTerm    ::= 'OUTPUT' TermId
TermId        ::= IRIREF
EventTypeId   ::= IRIREF
IRIREF        ::= '<' ([^<>"{}|^`\]-[#x00-#x20])* '>'
```

Listing 1: SEPN agent functionality specification language

3. SEPN FOR MONITORING SEE FEDERATION

Semantic Web Services (SWS) [5] have been identified as a promising technology to tackle the issues of interoperability and automation in the Internet of Services, through the usage of machine-processable descriptions of functional, non-functional and behavioral Web service aspects. A *Semantic Execution Environment* (SEE) is a middleware capable of operating over those descriptions in order to automate the SWS consumption lifecycle[3].

A generic architecture of a SEE is presented in Figure 3. A SEE is usually realized on top of a service-oriented paradigm, where specific broker services are responsible to enact specific usage tasks. Broker services are further orchestrated in order to provide meaningful SEE processes, such as: *Web service discovery* (discovers services functionally capable to fulfill user defined goals and preferences), *Web service invocation* (invokes a Web service by relying on the behavioral descriptions) and *achieving the complete Web service lifecycle* (comprised of a sequence of the previous two processes connected by a Web service selection step). The most notable SEE implementations are WSMX[4] and IRS-III[5].

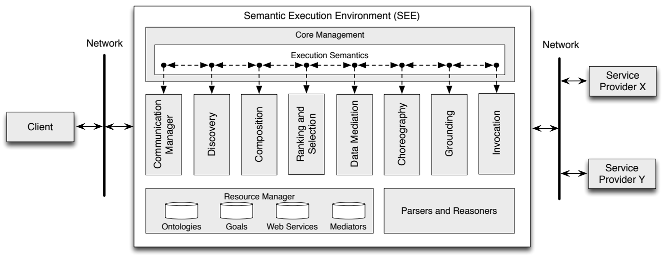

Figure 3: Architecture of a Semantic Execution Environment

As discussed in [6], there are scenarios which raise the need for a SEE instance to be able to communicate with other instances. This can be for scalability reasons, when a single instance is not able to handle all the knowledge (e.g., ontologies and service descriptions) and all the requests, or because different instances are established and operated by different authorities, and then federated so that users from each one can benefit from services provided by the others. In a federated SEE environment, the interactions among multiple "nodes" (i.e., SEE instances) can be quite complex. During service discovery, for example, a node can forward the user's

[2]The used EBNF notation is defined in Extensible Markup Language (XML) 1.1 Section 6 at `http://www.w3.org/TR/xml11`

[3]SEE is currently undergoing standardization process under OASIS SEE Technical Committee.

[4]`http://www.wsmx.org`

[5]`http://technologies.kmi.open.ac.uk/irs`

goal to other nodes, or decompose it into subgoals and forward these subgoals. A match may be identified through a composition of services registered at different nodes, and this composition can be registered as a new service. Therefore, not only discovery, but also invocation may involve multiple nodes, as it can imply the need to run an orchestration of services registered at different nodes.

Being built on top of a rich knowledge environment, SEE can certainly benefit from introducing event-based monitoring and reactivity support. In [9] we have shown that closing the knowledge-based control loop improves SEE in terms of auditing, run-time performance optimization, adaptation, and resilience to failures. This is even more important in a federated SEE environment, where events occurring at different nodes can be correlated, and reaction can be needed at a node which is not the source of the event. Besides detecting and reacting to failures of services or nodes in the federation, monitoring functionalities can be used to collect statistics, not only about services but also about SEE instances, which can then be used for ranking during discovery. If the invocation of a service needs to be performed through the SEE instance at which it is registered (e.g., because needed knowledge or functionalities are not available elsewhere), properties such as availability and response time of SEE instances become important in selecting, for example, which of the services that match a goal should be included in a business process. Such statistics can even be used for managing the federation itself, for example in deciding to which node a request should be forwarded. Therefore, monitoring events in the network of nodes and reacting to them can enable a certain level of adaptability and self-organization in a federation of SEE.

In this section we present an application of SEPNs to monitoring SEE and SEE federation. First we describe an ontology covering the domain, then we present an extended SEE architecture which integrates SEPNs in order to achieve monitoring and reactivity capabilities, and finally we discuss an example in which a SEE federation is monitored and appropriate actions are enacted at run-time.

3.1 An Ontology for Federated SEE Monitoring

A SEE represents a middleware solution capable of executing *processes* composed of *activities* mirroring Web service lifecycle tasks and performed by *agents* represented by SEE broker services. In order to successfully capture the behavior of federated SEE, a monitoring schema must capture those notions, genuinely coming from the domain of business processes monitoring.

The monitoring domain could be modeled and curated by relying on well-established technologies such as relational databases; we however choose to rely on an ontological representation to describe the domain of monitoring federated SEE. First of all, since SEE is a system which operates on top of ontologically represented data, the usage of ontologies in the core of the monitoring system provides a natural fit, enabling an end-to-end solution based on a homogeneous set of languages and tools, thus reducing implementation complexity and overall footprint of the solution. Moreover, ontologies allow richer constructs to represent data structures and, depending on the used formal theories, enable entailment of new knowledge. Last but not least, tools, query languages and repositories for ontologically represented data are getting mature enough for production deployments.

Figure 4 shows our ontology for monitoring federated SEE, which is a simplified version of the Core Ontology for Business Process Analysis (COBRA) [14]. In a nutshell, the ontology is built around the concept of `Event`, which represents a notification (property `occurredAt`) and the subsequent capture (property `receivedAt`) of an instantaneous change inside of the observed system. A set of `Data` instances can be associated to an `Event` instance, thus further refining its description. The observed system may perform an activity, which stands for an atomic unit of functionality (modeled by the concept `Activity`) spanning a time interval (property `interval`) bounded by events (property `event`) associated to the activity. An activity is performed by an agent (property `agent`) and it may also be associated with a set of `Context` instances and a set of `Property` instances. A process (represented by the concept `Process`) is a complex activity which orchestrates other activities (property `activity`). The concepts defined by the ontology core are domain independent, and as such can be used to record the behavior of virtually any business process.

In order to adjust the ontology to the federated SEE domain, we introduce further extensions. Concepts such as `Activity`, `Agent`, `Event`, `Process`, `Context`, and `Property` have been further refined to cover the domain of discourse. For example, `Process` is specialized to `InvokeWebService`, `DiscoverWebService`, and `AchieveGoal`, which cover SEE process definitions as explained at the beginning of section 3.

3.2 Architecture of Reactive Federated SEE

The integration of SEPN concepts into the SEE environment is performed at two levels. At the level of a single SEE instance, SEPN can answer to the needs for run-time SEE instance behavior monitoring, on top of which more advanced mechanisms can be built, e.g., ranking and selection of Web services according to exercised performance, or implementation of fail-safe procedures in the context of Web service invocation failures. Some of the mechanisms have already been demonstrated in Komazec et al. [9]. At the level of federation, an analogous approach can be used to monitor the behavior of the federated nodes in order to collect statistical data and adapt to the network changes (e.g., unavailability of a node must be immediately notified to all dependent parties in order to prevent further discovery and invocation of Web services hosted by that node).

Figure 5 depicts the architecture of reactive SEE. All components of the SEE instance have been instrumented in order to emit appropriate events according to the ontology presented in section 3.1. Enactment of the reactive behavior is devoted to the SEE Monitoring and Reactivity block, which is composed of two parts: `SEPN Environment` and `Action Handler`. The former represents an execution environment capable of running multiple SEPN instances while keeping intermediary results in the `Repository`. The latter is capable of interpreting events generated by running SEPNs in order to execute one of the following actions:

- *Repository Update.* This action enacts update over SEE instance repository content. In particular, ontologies and Semantic Web Service descriptions can be updated in order to adapt SEE instance behavior according to the encountered circumstances (e.g., Web service invocation failure or computation of service-

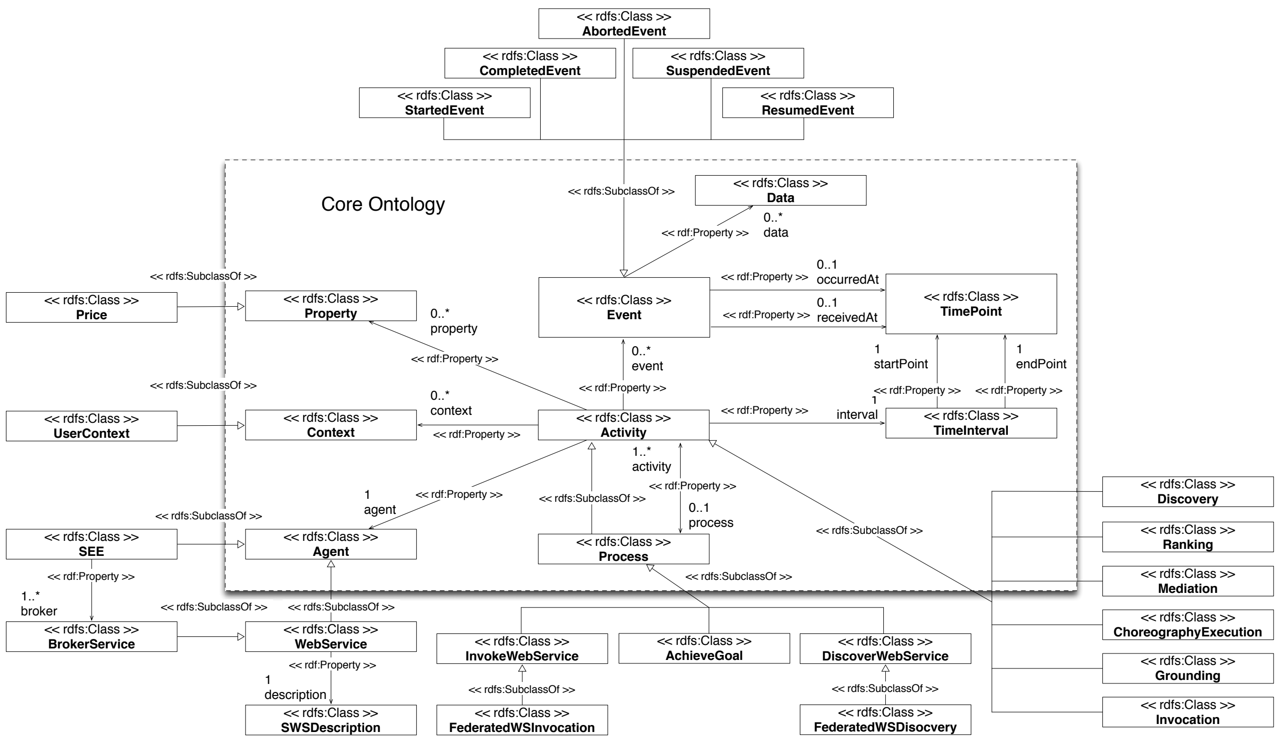
Core Ontology
<< rdfs:Class >> AbortedEvent
<< rdfs:Class >> CompletedEvent
<< rdfs:Class >> SuspendedEvent
<< rdfs:Class >> StartedEvent
<< rdfs:Class >> ResumedEvent
<< rdfs:SubclassOf >>
<< rdfs:Class >> Data
0..* data
<< rdf:Property >>
<< rdfs:Class >> Event
0..1 occurredAt
0..1 receivedAt
<< rdfs:Class >> TimePoint
<< rdfs:Class >> Price
<< rdfs:Class >> Property
0..* property
0..* event
1 startPoint
1 endPoint
<< rdfs:Class >> UserContext
<< rdfs:Class >> Context
0..* context
<< rdfs:Class >> Activity
1 interval
<< rdfs:Class >> TimeInterval
1..* activity
1 agent
0..1 process
<< rdfs:Class >> SEE
<< rdfs:Class >> Agent
<< rdfs:Class >> Process
1..* broker
<< rdfs:Class >> BrokerService
<< rdfs:Class >> WebService
1 description
<< rdfs:Class >> SWSDescription
<< rdfs:Class >> InvokeWebService
<< rdfs:Class >> AchieveGoal
<< rdfs:Class >> DiscoverWebService
<< rdfs:Class >> FederatedWSInvocation
<< rdfs:Class >> FederatedWSDisocvery
<< rdfs:Class >> Discovery
<< rdfs:Class >> Ranking
<< rdfs:Class >> Mediation
<< rdfs:Class >> ChoreographyExecution
<< rdfs:Class >> Grounding
<< rdfs:Class >> Invocation

Figure 4: An ontology for federated SEE monitoring

level statistics such as average invocation time).

- *Web Service Invocation.* This action enables invocation of a Web service registered at a SEE instance. This action provides a suitable means to access external systems (e.g., communicating with an external reputation manager) or to emit notifications to interested parties (e.g., sending emails to the administrators upon breaching a threshold).

In case of federated SEE deployment, the underlying network consists of multiple SEE instances sharing a repository used to store instance-wide functional and non-functional SWS descriptions, such as instance ranking levels and availability considerations. Events emitted by instances are processed by the `SEPN Environment`, which enacts the aforementioned activities over the selected instances and the federation repository.

3.3 An Example

In this section we present an example coming from the Enterprise Interoperability (EI) domain. This domain offers a variety of services, among which business process format transformation services play a significant role. These services target to transform documents between different formats such as ARIS Markup Language (AML)[6], EPC Markup Language (EPML)[7] and Scalable Vector Graphics (SVG)[8]. Figure 6 depicts a federation of SEE nodes where EI services are scattered across the nodes. More specifically, `SEE Node 1` has a service registered to transform EPML $\rightarrow$ AML, `SEE Node 2` supports a service capable of transforming AML $\rightarrow$ EPML, and `SEE Node 3` enables access to a service transforming EPML $\rightarrow$ SVG. If a client approaches `SEE Node 1` with a goal to perform an AML $\rightarrow$ SVG transformation, the node alone is not capable of responding, but the federation of SEE instances can compose a suitable service at the node which orchestrates an invocation of services registered at the other two nodes.

As depicted in Figure 6, the federation is monitored by relying on two SEPN instances. While `Ranking Management SEPN` computes statistics about the node behavior (i.e., computes average invocation time for the specific components and stores the values in the `Federated Repository`), the `Failure Management SEPN` listens for particular `AbortEvent` instances in order to decide about the availability of a SEE instance in case of invocation failures. The later network is composed of two agents: a `Filtering Agent`, which passes through only events satisfying requirements of being a SEE instance invocation failure, and a `Derivation Agent`, which builds up a `RepositoryUpdate` action stating that the SEE node description inside of the `Federated Repository` should be marked as unavailable. The unavailability of the node will prevent any further discovery and invocation attempts that depend on the node, until the node becomes available again.

Listing 2 shows a declarative specification of the `Derivation Agent` behavior. The `FROM` clause specifies a pattern describing failure (occurrence of a `seemon:AbortedEvent` instance) of a federation node to invoke another federation node with the intention to execute remote Web service invocation (relation to the instance `seemon:Federated WSInvocation`). The agent exploits the harvested data while

[6]AML is a trademark of IDS Scheer AG. The use of registered name and trademark in this paper does not imply that such names are free for general use.
[7]`http://www.mendling.com/EPML`
[8]`http://www.w3.org/Graphics/SVG`

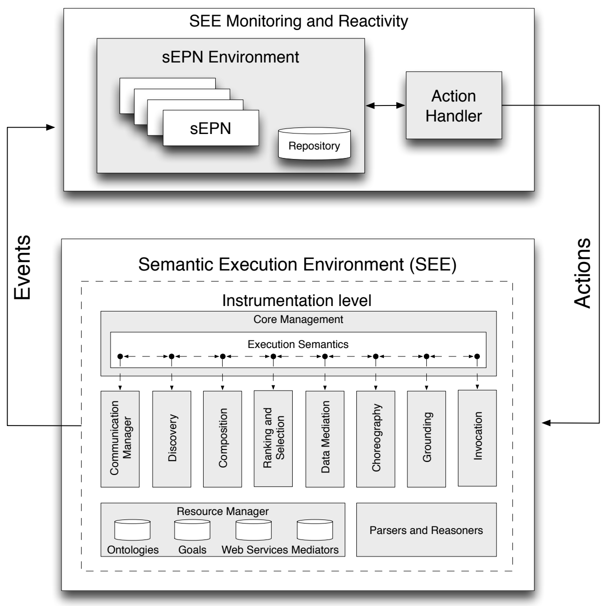

Figure 5: Architecture of reactive SEE

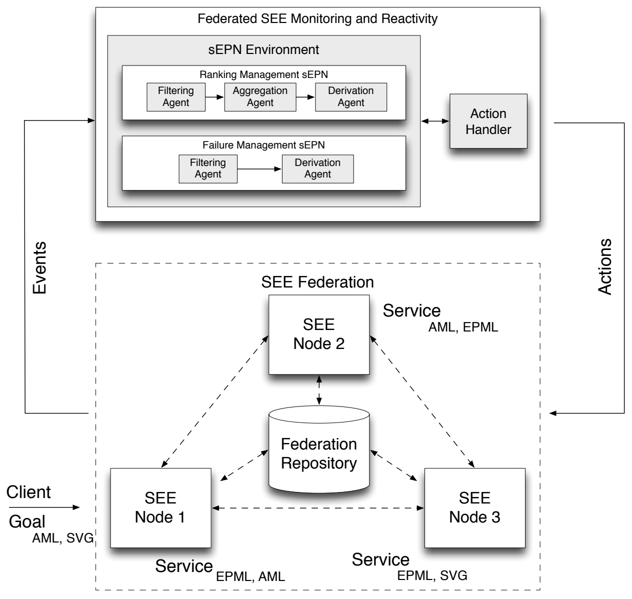

Figure 6: Enterprise Interoperability use case

executing the `DERIVE` clause which constructs a new event (instance of `seemon:UpdateRepo`) to be consumed by the `Action Handler` and which marks the failed node unavailable for further invocations.

```
PREFIX
 sepn: <http://www.sepn.example/>
 seemon: <http://www.seemon.example/>
 rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
 xsd: <http://www.w3.org/2001/XMLSchema#>
 fed: <http://www.seefederation.example/>

INPUT TERMINAL
 <http://www.sepn.example/InputTerminal>
 seemon:AbortedEvent

OUTPUT TERMINAL
 <http://www.sepn.example/OutputTerminal>
 seemon:UpdateRepo

DERIVER
 INPUT  <http://www.sepn.example/InputTerminal>
 OUTPUT <http://www.sepn.example/OutputTerminal>
 DERIVE {
  sepn:UpdateRepoX rdf:type seemon:UpdateRepo.
  sepn:UpdateRepoX seemon:repoId sepn:federatedRepo.
  sepn:UpdateRepoX seemon:update ?seeNode.
  ?seeNode fed:availabilityNFP "false"^^xsd:boolean.
 }
 FROM {
  ?abortedEvent rdf:type seemon:AbortedEvent.
  ?process seemon:event ?abortedEvent.
  ?process rdf:type seemon:FederatedWSInvocation.
  ?process seemon:agent ?seeNode.
  ?seeNode rdf:type seemon:SEE.
 }
```

Listing 2: Specification of a failure processing derivation agent

4. RELATED WORK

In contrast to our previous work [9], where we relied on Abstract State Machines for defining event detection patterns and accompanying reactions, in this work we focus on Event Processing Networks, as a more flexible approach to digest event streams.

ETALIS represents one of the recent academic approaches (Anicic et al. [1]) to tackle the issue of logic-based event processing. It provides a rich set of system operands and predicates for expressing complex event patterns executed on top of event-driven backward chaining rules executed in data-driven fashion. Recently, ETALIS has been enriched with the Event Processing SPARQL (EP-SPARQL)[9] language, which enables the specification of event patterns in a SPARQL-like syntax and performs processing over RDF event streams. ETALIS and SEPN represent two complementary solutions. While the former concentrates on the core event processing mechanism, the latter aims at providing a Semantic Web-compliant event processing solution with the level of flexibility needed for application at a larger scale (e.g., federated SEE environment).

C-SPARQL (Barbieri et al. [2, 3]) represents a recent attempt to adapt SPARQL to the continuous query processing model. In essence, it introduces a notion of RDF stream which is an ordered sequence of RDF triples associated with timestamps, on top of which it builds time windowing support. In contrast to the stream processing embraced by C-SPARQL, our work targets event processing. While sharing a lot of commonalities (i.e., real-time data analysis) the two approaches emphasize different abstraction levels: stream processing resides at a lower abstraction level (e.g., in the case of C-SPARQL it is a stream of RDF triples), while event processing assumes a higher abstraction level by identifying stream sections which contribute to an event description (e.g., temporal, geo-spatial, contextual, payload, etc.). The

[9]More information can be found at `http://code.google.com/p/etalis`

presence of the event dimensions enables powerful mechanisms to correlate events, calculate causality relationships, and apply trend and spatiotemporal patterns (e.g., moving in a constant direction, etc.).

The work presented in this paper shares some commonalities with the solutions providing Event-Condition-Action (ECA) rule frameworks on top of Semantic Web technologies. In particular, the RDF Triggering Language (RDFTL) presented by Papamarkos et al. [13] provides reactive behavior over RDF data stored in an RDF repository. Similarly to our approach, RDFTL operates over RDF graphs from which it extracts triples or their parts according to the specified criteria. RDFTL expresses RDF selections and constraints over path expressions built on top of XPath, while our solution uses SPARQL graph pattern-based constructs which provide more expressive power. Since RDFTL operates on top of an RDF repository, the supported event types are falling into the range of common database triggers (i.e., insertions and deletions of triples), while the action part also targets activities over stored data (i.e., insertions, deletions and updates of triples). Our solution is domain independent, and as such can be applied to various situations (including detection of RDF repository activities).

Using ontologies to represent data collected during monitoring of a distributed system has not been a preferable choice so far: the scalability of data repositories used to be limited, the expressivity and flexibility of query languages constrained, and overall the technology was rather immature to provide suitable support for industrial-strength deployments. One of the first distributed system areas to embrace ontologies as a suitable solution to organize and process monitored data was Grid computing (Nazir et al. [12], Yu et al. [22], and Truong et al. [17]). Unambiguous and formal descriptions of resources and their consumption can contribute to reconcile heterogeneities between the grid nodes and scale-up grid usage tasks (e.g., resource discovery and consumption).

Considerable work in the area of Semantic Web Service monitoring and fault management has been reported in Vaculin et al. [18, 19, 20] and Wiesner et al. [21]. The authors provide descriptive event and exception event taxonomies capable of capturing various phases and exceptional situations during an OWL-S service execution. On top of them, the authors build a composite event detection mechanism grounded in event algebra and providing exception handling and compensation-based recovery mechanisms. However, their work concentrates on OWL-S, while our solution targets a generic SEE framework; moreover, regarding monitoring the reported work covers only part of the overall SWS usage lifecycle (i.e., execution of SWS inside OWL-S Virtual Machine) while our approach covers the whole lifecycle. This enables us to reason over the complete knowledge gathered inside of a federated SEE environment. Rather than focusing only on instantaneous temporal constructs (i.e., events) to convey information about the service execution progress, our approach focuses also on the spanning temporal constructs (i.e., activities) bounded by events as a more natural way to describe phenomena inside SEE, which gives more expressivity when it comes to correlating various events and activities.

The ontology described in section 3.1 represents a simplified version of the Core Ontology for Business Process Analysis (COBRA) developed by Pedrinaci et al. [14]. COBRA represents an attempt to foster automation in analyzing business processes by providing an expressive, elaborate and flexible framework built on top of the commonly accepted notions from the domain of Business Process management (BPM) such as `Activities`, `Processes`, `Events`, and `Agents` related to each other through a set of `Roles`. Our ontology represents a lightweight version of COBRA, where a set of simplifications have been applied (e.g., exclusion of `Roles` support or simplification of the `Agent` hierarchy description), in order to reduce the computational overhead and align more towards capturing knowledge in a federated SEE environment.

5. CONCLUSIONS AND FUTURE WORK

In this paper we presented Semantically-enhanced Event Processing Networks (SEPNs): an approach, built on top of the SPARQL query language, to specialize the conceptual model of Event Processing Networks while declaratively describing the behavior of Event Processing Agents. We applied our approach to extend the architecture of a Semantic Execution Environment, integrating event-based monitoring and reactivity support in order to facilitate implementation of adaptation and self-management strategies. We proposed an ontology governing the concepts and relationships related to the domain of monitoring federated SEE, and discussed an example in which a SEE federation is monitored and appropriate actions are enacted at run-time.

We see future improvements of our SEPN approach in several directions. First, we plan to thoroughly analyze how existing RDF stream processing solutions (such as C-SPARQL) can be embraced and further refined in order to answer to event processing challenges. More specifically, functions peculiar to the event processing paradigm (such as threshold, trend and geo-spatial computations) should be introduced in order to enable elegant and focused, yet expressive support to define the behavior of Event Processing Agents. Second, in order to prepare SEPN approach for large-scale applications, we plan to introduce a formal model in which all constituent parts of the SEPN (i.e., event producers and consumers, event channels and event processing agents) will have a precise and machine processable characterization of their capabilities and properties. Such a model will help us in taming the interoperability and scalability challenges that an event processing application can encounter in diverse and large deployment scenarios. Third, we seek to improve the performance of RDF graph pattern detection methods, which represent a crucial factor for efficient SEPN applications. We believe that the integration of existing many-pattern/many-data evaluation methods (e.g., the RETE algorithm [8]) with knowledge representation inference rules (e.g., RDFS [4]) could yield performance improvements in contrast to the existing approaches tackling the issue of continuous query answering over rapidly changing data.

6. ACKNOWLEDGMENTS

The paper is mainly based on work performed in the project COIN (EU FP7 Project 216256; www.coin-ip.eu) funded by the European Community within the IST Programme of the 7th Framework Research Programme. The authors also thank the contribution from other partners in the COIN consortium.

7. REFERENCES

[1] D. Anicic, P. Fodor, R. Stuhmer, and N. Stojanovic. Event-Driven Approach for Logic-Based Complex Event Processing. In *Proceedings of the 2009 International Conference on Computational Science and Engineering - Volume 01*, CSE '09, pages 56–63, Washington, DC, USA, 2009. IEEE Computer Society.

[2] D. F. Barbieri, D. Braga, S. Ceri, E. Della Valle, and M. Grossniklaus. C-SPARQL: SPARQL for continuous querying. In *WWW '09: Proceedings of the 18th International Conference on World Wide Web*, pages 1061–1062, New York, NY, USA, 2009. ACM.

[3] D. F. Barbieri, D. Braga, S. Ceri, and M. Grossniklaus. An execution environment for C-SPARQL queries. In *EDBT '10: Proceedings of the 13th International Conference on Extending Database Technology*, pages 441–452, New York, NY, USA, 2010. ACM.

[4] D. Brickley and R. Guha. RDF Vocabulary Description Language 1.0: RDF Schema. W3C Recommendation, 2004.

[5] J. Cardoso and A. P. Sheth. *Semantic Web Services, Processes and Applications (Semantic Web and Beyond: Computing for Human Experience).* Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

[6] D. Cerri and S. Komazec. Towards a Federation of Semantic Execution Environments. In *Proceedings of the 1st International Future Enterprise Systems Workshop (FES-2010)*, volume 659 of *CEUR Workshop Proceedings*, 2010.

[7] O. Etzion and P. Niblett. *Event Processing in Action.* Manning Publications Co., Sound View Ct. 3B, Greenwich, CT 06830, 2010.

[8] C. L. Forgy. Rete: A Fast Algorithm for the Many Pattern/Many Object Pattern Match Problem. In P. G. Raeth, editor, *Expert systems*, pages 324–341. IEEE Computer Society Press, Los Alamitos, CA, USA, 1990.

[9] S. Komazec and F. M. Facca. Towards a Reactive Semantic Execution Environment. In *OTM '09: Proceedings of the Confederated International Workshops and Posters on On the Move to Meaningful Internet Systems*, pages 877–887, Berlin, Heidelberg, 2009. Springer-Verlag.

[10] 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.

[11] F. Manola and E. Miller. RDF Primer. W3C Recommendation, 2004.

[12] F. Nazir, H. F. Ahmad, H. A. Burki, T. H. Tarar, A. Ali, and H. Suguri. A Resource Monitoring and Management Middleware Infrastructure for Semantic Resource Grid. In P. Herrero, M. S. Perez, and V. Robles, editors, *Scientific Applications of Grid Computing*, volume 3458 of *Lecture Notes in Computer Science*, pages 188–196. Springer Berlin / Heidelberg, 2005.

[13] G. Papamarkos, A. Poulovassilis, and P. T. Wood. RDFTL : An Event-Condition-Action Language for RDF. In *Proceedings of the 3rd International Workshop on Web Dynamics (in conjunction with WWW2004)*, pages 223–248, 2004.

[14] C. Pedrinaci, J. Domingue, and A. K. Alves de Medeiros. A Core Ontology for Business Process Analysis. In S. Bechhofer, M. Hauswirth, J. Hoffmann, and M. Koubarakis, editors, *Proceedings of the 5th European Semantic Web Conference*, volume 5021 of *Lecture Notes in Computer Science*, pages 49–64, Berlin, Heidelberg, 2008. Springer-Verlag.

[15] E. Prud'hommeaux and A. Seaborne. SPARQL Query Language for RDF. W3C Recommendation, 2008.

[16] A. Sheth, C. Henson, and S. S. Sahoo. Semantic Sensor Web. *IEEE Internet Computing*, 12(4):78–83, 2008.

[17] H.-L. Truong, S. Dustdar, and T. Fahringer. Performance metrics and ontologies for Grid workflows. *Future Gener. Comput. Syst.*, 23(6):760–772, 2007.

[18] R. Vaculín and K. Sycara. Monitoring execution of OWL-S web services. In *Proceedings of OWL-S: Experiences and Directions Workshop, 4th European Semantic Web Conference*, June 2007.

[19] R. Vaculín and K. Sycara. Specifying and Monitoring Composite Events for Semantic Web Services. In *ECOWS '07: Proceedings of the 5th European Conference on Web Services*, pages 87–96, Washington, DC, USA, 2007. IEEE Computer Society.

[20] R. Vaculín, K. Wiesner, and K. Sycara. Exception Handling and Recovery of Semantic Web Services. In *ICNS '08: Proceedings of the 4th International Conference on Networking and Services*, pages 217–222, Washington, DC, USA, 2008. IEEE Computer Society.

[21] K. Wiesner, R. Vaculín, M. Kollingbaum, and K. Sycara. Recovery Mechanisms for Semantic Web Services. In R. Meier and S. Terzis, editors, *DAIS '08: Proceedings of the 8th IFIP WG 6.1 international conference on Distributed applications and interoperable systems*, volume 5053 of *Lecture Notes in Computer Science*, pages 100–105, Berlin, Heidelberg, 2008. Springer-Verlag.

[22] Y. Yu and H. Jin. An Ontology-Based Host Resources Monitoring Approach in Grid Environment. In W. Fan, Z. Wu, and J. Yang, editors, *Advances in Web-Age Information Management*, volume 3739 of *Lecture Notes in Computer Science*, pages 834–839. Springer Berlin / Heidelberg, 2005.