# Monitoring Web Service Event Trails for Business Compliance

Emmanuel Mulo, Uwe Zdun, Schahram Dustdar
*Distributed Systems Group*
*Institute of Information Systems*
*Vienna University of Technology*
*Vienna, Austria*
{*e.mulo,zdun,dustdar*}@*infosys.tuwien.ac.at*

***Abstract*—Organizations today are required to adhere to a number of compliance concerns from laws, regulations and policies. Compliance is achieved through defining and implementing so-called controls within the organizations' business processes. Organizations that build their systems based on the process-driven SOA paradigm realize business processes through invocation of services to handle the different process activities. We propose an approach for monitoring business compliance in service-oriented systems—we represent a service invocation as an event, enabling us to map business process activities into trails of events that make up compliant business processes. The event trails guide the creation of rules, which are leveraged by complex event processing techniques to monitor business processes for compliance. A case study from the telecommunications sector is used to demonstrate how we achieve compliance detection. In this case study, we use our approach to identify violations of licensing compliance requirements in the business process of a multimedia service provider.**

***Keywords*-Web services, complex event processing, business compliance monitoring**

## I. Introduction

Organizations today are required to adhere to a vast number of compliance concerns, for example, acts of law from governments, regulations drawn up by regulatory authorities, and the organization's own internal policies and procedures. Non-adherence to compliance concerns may have consequences such as loss of credibility, financial losses, and possibly having legal actions taken against the organization. These concerns should be taken into consideration when designing and implementing information systems in organizations. In particular, the compliance concerns should translate into artifacts or controls, embedded within the organization's information systems, that aim for prevention (or detection) of compliance violations.

Information systems, based on the Service-Oriented Architecture (SOA) paradigm, are designed to have different functions encapsulated as services. A process-driven SOA [1] introduces a process engine that orchestrates these services to perform the different activities that make up a business process. In other words, a business process execution is realized as the invocation of several service operations, each performing a specific task in the process. In large-scale process-driven SOA systems [1], [2], multiple business process instances are executed and coordinated on multiple process engines. All process instances are realized through invoking operations from a pool of services that are within and sometimes beyond the boundary of an organization.

During execution of these processes, a number of issues may occur, e.g., system failures, process failures, service failures, or human errors, that can result in the processes not executing as expected, and as a result not adhering to the required compliance concerns. Monitoring business processes as they execute provides organizations with feedback that helps to rectify system issues, redesign business processes, or alert responsible individuals to solve the problematic issues. In this work, we are interested in feedback related to violation of compliance concerns. Since a business process is realized as a combination of multiple service invocations, it follows that generating service invocation events enables us to monitor the process as it executes. In a large-scale process-driven SOA system, a large volume of events would be generated.

A number of studies of monitoring solutions [3], [4], [2], [5] propose an external component to which events are sent. In these monitoring solutions events are recorded in audit logs (files) that can later be analyzed to identify anomalies in system behavior. Other research proposes monitoring events in real-time using complex event processing (CEP) techniques [6], [7]. With CEP, event-based systems correlate and aggregate events as they occur, to discover and respond to certain event patterns [7].

We propose an approach for monitoring service-oriented systems, to detect violations of compliance concerns during execution of an organization's business processes. In our approach, we represent a service invocation as an event – we can then take an existing business process that is considered compliant, break it down into individual business activities, and map these activities into trails of service invocation events. These event trails drive the creation of rules and queries to identify anomalous process executions, that may have resulted from compliance violations. We have evaluated our approach through a case study from the telecommunications sector. In addition, we have analyzed the performance and scalability of our approach by running test scenarios realistically mimicking a large-scale process-driven SOA with event monitoring.

The rest of this paper is structured as follows. First, we give an example to motivate the approach we present. Section III gives background information concerning compliance and complex event processing. Next, Section IV explains our proposed approach, and Section V goes into details concerning the case study that we use to demonstrate our approach. Section VI compares our work to the related work, and finally Sections VII and VIII present discussions and a conclusion to this work.

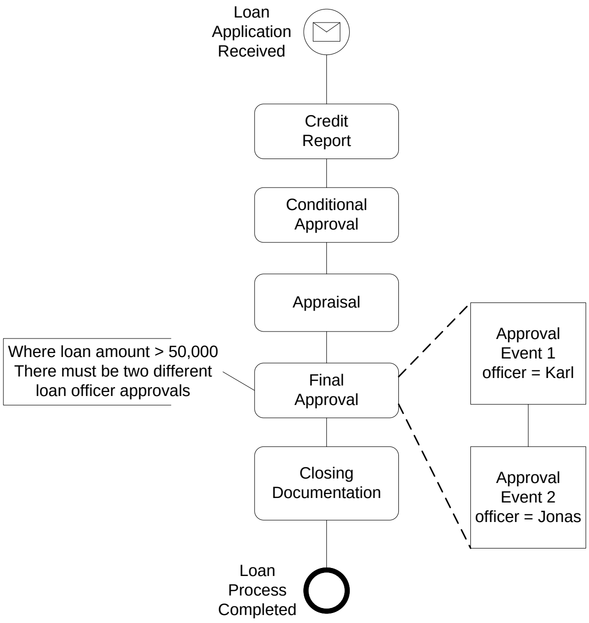

Loan Application Received
Credit Report
Conditional Approval
Appraisal
Where loan amount > 50,000 There must be two different loan officer approvals
Final Approval
Approval Event 1 officer = Karl
Approval Event 2 officer = Jonas
Closing Documentation
Loan Process Completed

Figure 1. Loan process compliance

## II. Motivating Example

We explain our proposed approach using a simple motivating example of a loans issuing procedure in a bank. A compliance control in the bank states that no loan officer is allowed to single-handedly approve loans greater than EUR 50,000. A simplified business process of the loans procedure is shown in Figure 1.

The *final approval* activity of the process is implemented as a service whose approval operation needs to be invoked by two different loan officers (each invocation counts as a separate event), if the loan amount is greater than EUR 50,000. The loan approval process is compliant if these conditions are fulfilled during execution of the final approval activity.

With these conditions, the event trail for the *final approval* business activity should consist of two service invocation events in cases where the loan amount is greater than EUR 50,000. The event trails drive the creation of rules/queries that monitor for compliance at runtime. For example, in Figure 1 we identify the flow of events (approval event 1 followed by approval event 2) and can thus define what pattern to search for. We also identify conditions (loan amount greater than EUR 50,000) that are used in predicates for event processing. These patterns and predicates fit naturally into the kind of queries used in CEP techniques to process events.

## III. Background

In this section, we consider some background information on the two themes, compliance in business organizations and complex event processing, that are brought together in this paper.

### A. Compliance in Organizations

Compliance concerns in an organization are related to risks the organization and its stakeholders face in achieving their mission. The concerns guide organizations as to what measures should be taken to prevent occurrence of these risks, e.g., Section 404 of Sarbanes-Oxley Act (SOX) requires public companies to annually assess and report on the design and effectiveness of internal control over financial reporting. However, organizations are not given guidelines on *how* to implement these measures.

During risk management exercises, organizations decide on the *controls*, i.e., risk-reducing measures taken [8], and their implementation. Preventative controls aim to avoid the risks from occurring, whereas detective controls warn of the occurrence of risks . However, organizations need not invent all controls from scratch. A number of established norms or standards define and describe standard controls that must be adapted and implemented in an organization to prevent certain risks. The Control Objectives for Information and related Technologies (COBIT) [1] framework, for example, describes control objectives that an organization can use as a guide in making choices about what controls to implement. These norms and standards are fairly generic and abstract, and must be mapped to a concrete systems implementation.

In many cases, unfortunately, this mapping does not follow a generic strategy or guideline, and hence business compliance is reached on a per-case basis. That is, organizations use ad hoc, hand-crafted solutions for specific rules they must comply with. This usually means that, for each set of compliance concerns to be addressed, a separate project is started. Each separate project develops an individual, custom solution for the particular compliance concern to be addressed. From a software architecture perspective, this is undesirable because such solutions are often:

- hard to *maintain* because ad hoc, hand-crafted solutions usually do not follow a clear architectural concept throughout the SOA;
- hard to *evolve* or change because usually the ad hoc, hand-crafted solutions lead to tangled code that is spread over the systems and has many dependencies to other components that are hard to resolve;
- hard to *reuse* because ad hoc, hand-crafted solutions often involve special purpose code added into the systems at several places;
- hard to *understand* because tangled code added at several places offers no adequate separation of concerns.

As a result, it becomes quite a task to ensure compliance to a given set of rules and regulations, and to keep up with constant changes in regulations and laws. In this paper, we focus on a specific kind of compliance concerns: those that can be observed as runtime events. For these kind, we apply our approach as a generic strategy to monitor compliance. This approach is based on observing event trails via complex event processing – which is explained as a background technique in the next section.

[1] www.isaca.org/cobit/

### B. Complex Event Processing

Complex Event Processing (CEP) is a set of tools and techniques for analyzing and controlling a complex series of interrelated events [9]. These techniques have a number of application areas, including policy enforcement and regulatory compliance [9], [7].

Events represent the occurrence of an activity within a system. They originate from a number of sources, e.g., RFID tags, network traffic data, and enterprise application components, and they may contain information, e.g., the event source, or time of occurrence. This information enables us to analyze the events and determine how they relate to each other.

Domain specialists are interested in events of significance in their domain. In some cases we are not able to directly observe such special events because they occur as a combination of a number of other events. However, through the use of event pattern languages (EPLs), it is possible to aggregate what are termed low-level events into complex (high-level) events. Low-level events are not an abstraction of other events, and do not have semantic significance on their own within a specific domain, whereas complex events are an aggregated abstraction of a number of other low-level and/or complex events [9], [10]. CEP techniques filter and correlate low-level events to yield other higher-level, more semantically significant events [11], i.e. special events.

In our work, we draw parallels between the special, domain-specific events of interest and complex events, and between service invocation events (invoked from executable business processes) and low-level events.

## IV. Event-based Compliance Detection

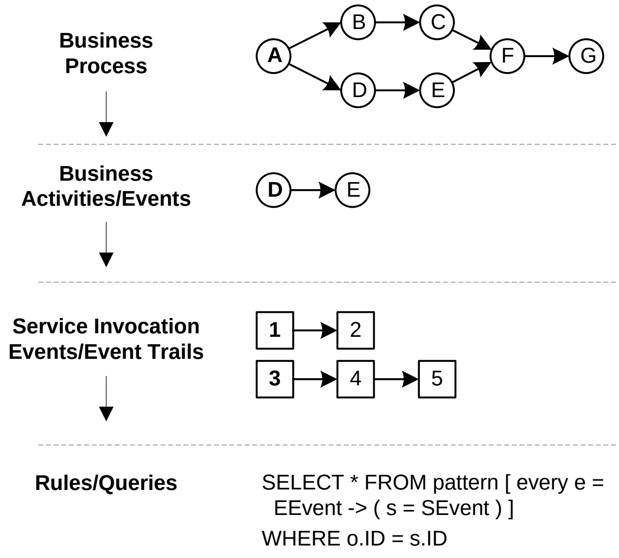
### A. Overview of Compliance Detection Approach

In process-driven service-oriented systems, services are orchestrated in order to realize a business process. Each service executes a particular function of the entire business process. From the perspective of business operations, however, the business process is realized by composing a number of business activities.

For business operations, which are required to be compliant, we are interested in monitoring the flow of business activities. Since business activities are realized by one or more service invocations, we are able to monitor these activities by monitoring the service invocations.

To achieve this, we represent the invocation of each service as an event with details like, the operation invoked, and the parameters passed to the operation. These events are gathered by either instrumenting a service or wrapping its interface with event emitting code. Consequently, we are able to identify the service invocation *event trails* that realize each business activity. These event trails guide us in creation of rules and queries to identify and monitor business activities. As a result, we are able to determine whether compliance is violated during the execution of business processes. Figure 2 illustrates this transition from business processes to compliance detection.

In Figure 3 we present a model illustrating the relationship between events, business activities and business processes. Business activities, which are the domain-specific events of interest, are represented by business events. Each business event may be related to at least one service invocation event and any number of other business events. Each service instance is related to a number of service invocation events. This is because a service might consist of several operations, each representing a different event. We choose to have a one-to-one mapping of events to operation invocations in order to identify the execution trace of a business activity.


Business Process
A
B
C
D
E
F
G
Business Activities/Events
D
E
Service Invocation Events/Event Trails
1
2
3
4
5
Rules/Queries

```
SELECT * FROM pattern [ every e =
  EEvent -> ( s = SEvent ) ]
WHERE o.ID = s.ID
```

Figure 2. Approach from business processes to compliance detection

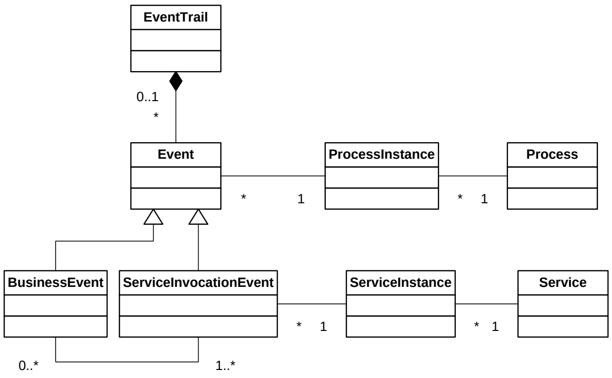


Figure 3. Event model

### B. Mapping Business Activities to Event Trails

A service invocation event contains parameters such as when it occurred, the source of the event, and operations (along with parameter values) invoked on the service. Service invocation event trails are essentially sequences of events that represent a single business activity. At runtime, these trails are identified by analyzing the parameters of events within monitoring (event) streams. These parameters can be said to leave a “trail” that relates a number of different events together into a single business activity. It is with these parameters that we derive queries that monitor

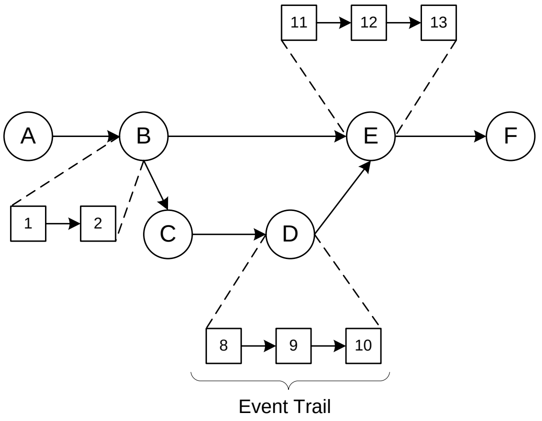
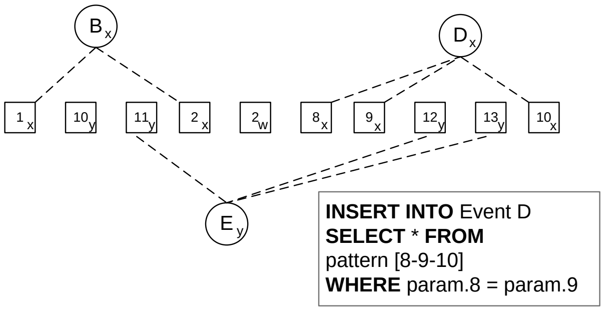


Figure 4. Business process graph and event trails



```
INSERT INTO Event D
SELECT * FROM
pattern [8-9-10]
WHERE param.8 = param.9
```

Figure 5. Rules to correlate event trails

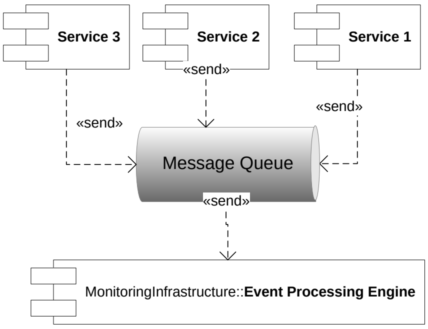


Figure 6. Monitoring infrastructure in SOA

business activities and, ultimately, compliance.

Consider Figure 4 that shows a subgraph of a business process, we have two possible flows; A-B-E-F or A-B-C-D-E-F. Each node represents a business activity. However, each business activity may be realized by multiple service invocation events. For example, activity D could be realized by the trail of events 8-9-10. Similar trails of events are identified for each business activity in the business process.

### C. From Event Trails to CEP Queries/Rules

Event trails identified from business activities drive the creation of event processing rules and queries. An event monitoring system has a sequential view of all the events as they arrive at its interface. As it executes, a single business process emits service invocation events from the individual business activities. However, in a large scale SOA, multiple instances of different business processes execute concurrently, resulting in the generation of an interwoven sequence of events.

Using event trails, we are able to define queries (see inset of Figure 5) that a CEP engine uses to correlate events and identify which business activities belong to which process instance. For example, the query shown searches for a business activity D from a stream of events. The event trail 8-9-10 matches activity D of a process instance (we use the subscript *x* to indicate that these events are in the same process instance). Correlation of events is performed through comparison of their parameters. Within the same stream in Figure 5, there are other events that belong to different process instances and are correlated into different business activities.

With the business activities identifiable, we are able to monitor the flow of a business process at runtime. As a result we can detect whether or not there have been any violations of compliance.

### D. Monitoring Infrastructure

In order to demonstrate this approach we design a simple monitoring infrastructure consisting of three components viz. a prototype of an event processing server, an event message queue, and web services. The integration of all components is illustrated in Figure 6.

A CEP engine is embedded within the event processing server (we use the Esper CEP engine [12]). At runtime, a single instance of the engine is created and configured with event types and event processing rules/patterns. Each rule may have a listener attached to it, that is invoked whenever the rule matches a pattern. Within the listeners, one can program actions that are executed when the listener is invoked, e.g., the listener can be programmed to update a user interface about the occurrence of an event. A number of worker threads receive events from the event message queue, and feed them into the CEP engine for processing.

The event message queue provides a bridge between the event processing server and the web services. The queue allows for multiple clients to send messages without overloading the event processing server. In our implementation we use the ActiveMQ [13] messaging provider as a message queue.

The last component is the web services. In a process driven SOA, a number of services are required to fulfill the different activities of a business process. Whenever they are invoked, these services send events to the event processing server. In our tests, however, we simulate the service invocation events of the business processes described in our case study in section V.

## V. Case Study: Applying the Solution

In this section, we apply our approach to the multimedia streaming business process. The process is a scenario of advanced telecom services offered by a Mobile Virtual

Network Operator (MVNO). MVNOs provide value-added services to users by accessing and aggregating various facilities from other content providers. In our scenario, multimedia streams are provided to mobile phone users through a special web service, the WatchMe service. The user's access rights vary based on licensing options (pay per use or time-based) they choose when paying for stream access. After paying, a user has access to a media stream for as long as their access rights are in accordance with the license selected. In the scenario, an MVNO is interested in monitoring the multimedia streaming process for adherence to the licenses, as well as detection (and reporting) of violations and anomalous events. We evaluate the approach by simulating the MVNO's business process and monitoring events with a simple event monitoring infrastructure.

### A. *Multimedia Streaming Business Process*

The flow of the business process is modeled in Figure 7. The detailed flow of the business process is as follows;

1) The user contacts the WatchMe service to access a media stream of interest.
2) The user enters search criteria which is submitted to the WatchMe service. The service in turn contacts a number of providers, with whom the MVNO has agreements, to check whether they have media streams matching the user's search criteria.
3) All search results from different providers are aggregated and provided to the user. The results contain parameters, e.g., type and language of the (audio) media streams.
4) The user selects a particular stream and is provided with more details, e.g., a description and a preview. In addition, the user is provided the costs/price and relevant license plan of accessing the stream.
5) The user selects a license plan and confirms payment.
6) Once the WatchMe service receives confirmation of payment, the stream is available to the user. Whenever the user accesses the media stream, the WatchMe service provides an access URL from a particular provider.
7) Each time the user accesses the media stream, the combination of the user identification and the stream URL is checked to ensure that license agreements are not violated.

### B. *Multimedia Streaming Event Trails*

Each activity in the business process is analyzed to determine which service invocations are required to realize business activities. We map the business activities to trails of service invocation events as shown in table I. Each service invocation event has a one-to-one mapping with an operation provided by a service. For example, in table I the activity *Receive media stream request* is realized through invocations of the *playMediaSteam* and *accessMediaStream* operations of the *WatchMeMediaProviders* service as shown in Figure 7.

We use this mapping to create relevant rules and queries required for event pattern identification, and by extension, compliance violation checking. In this case, we consider that the MVNO does not want users accessing media they have not paid for, and that is outside of the licensing plan the user has chosen.

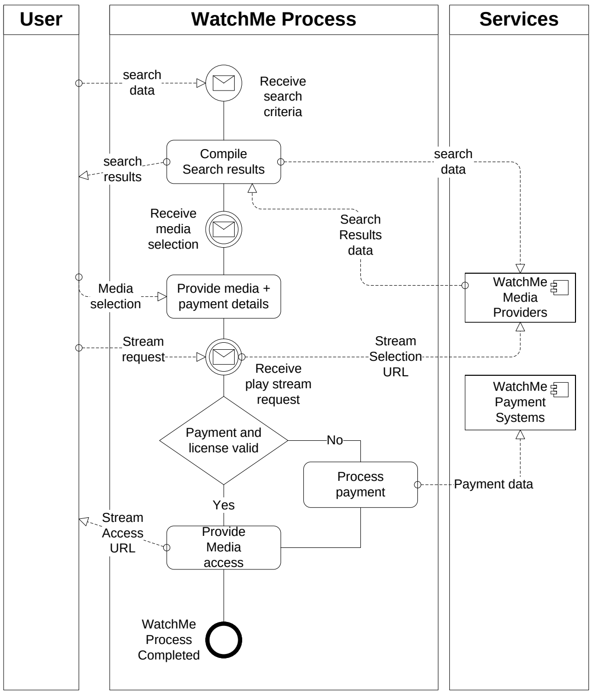


Figure 7. Multimedia stream access business process

Table I
MAPPING BUSINESS ACTIVITIES TO SERVICE INVOCATION TRAILS

| **Business Activities** | **Service Invocations** |
|---|---|
| Receive search criteria | searchMediaStream |
| Contact multiple providers | searchAudioStream - searchVideoStream |
| Receive user media selection | |
| Offer media details + payment request | |
| Receive media stream request | playMediaStream - accessMediaStream |
| Confirm payment | placeMediaOrder |
| Stream media | |

Listing 1 shows a query derived from the pattern of event trails in table I. On Lines 1–4 we search for a trail of service invocation events corresponding to the business activity *Receive media stream request*. Line 5 is a condition that checks whether the attempt to play the stream is accepted according to the license agreements, in this case whether one is attempting to play a media stream after the end date of the order in a time-based licensing plan. In this way we have run a compliance check on the licensing plan of the business process.

In our demonstration infrastructure, one can attach a listener to the query statement of Listing 1. If the query is matched during event stream processing, the listener is invoked. For our tests we only make a simple console log indicating that such an event–violation of compliance–occurred. However, this information could also be passed on to a more sophisticated dashboard for visualizations or further processing.

```
SELECT * FROM pattern [
every orderEvent(orderType='TIME_BASED') ->
(every playMediaEvent(orderID=orderEvent.orderID) ->
accessEvent(orderID=playMediaEvent.orderID)) ]
WHERE accessEvent.creationTimeStamp > orderEvent.endDate
```

Listing 1. Queries/Rules for Compliance Detection

### C. Evaluation

In this section we present the evaluation of our approach. We give an overview of the test environment and procedures, and then present results from test runs.

A dedicated server computer with an Intel Xeon 3.20GHz CPU and 2GB RAM is used to execute the tests. The server is installed with Ubuntu Linux 9.04 Server Edition. All three components for our test scenario are installed on the server. For the message queue component, we install ActiveMQ 5.2.0 and configure it to create a single message queue providing point-to-point message delivery. The event processing server uses the Esper 2.1.0 component as the embedded event processing engine. Finally, the web services simulation component (we code ourselves) creates multiple threads, each simulating a business process instance and submitting events to the message queue.

Our evaluation measures the performance of our monitoring infrastructure to give hints about the scalability of the approach. We consider throughput (rate of event processing) of the event processing server as a performance measure. We simulate three different test batches:

- a single business process (pay per use license scenario) with one rule to match the process pattern,
- two business processes with one rule to match the pattern of only one of the processes – the second process (time based license scenario) provides noise to observe whether there is any change in performance,
- two business processes with two rules, each rule matching one of the process patterns.

For each batch, we vary the number of threads (concurrent business process instances) executed by the web services simulation component. This way we observe the performance of the event processing server under different loads. For each fixed load, the tests are run 10 times and the average from all the runs is used to plot the graphs.

The processing time of the server is measured from the time the first event is submitted to the CEP engine until the last invocation of a listener. The processing time and the total number of events processed determine the event processing rate.

Figure 8 shows the data from the three test batches. Note that the x-axis shows concurrent process instances *per* business process. This means that in the scenarios with two business processes, the total number of concurrent process instances is doubled.

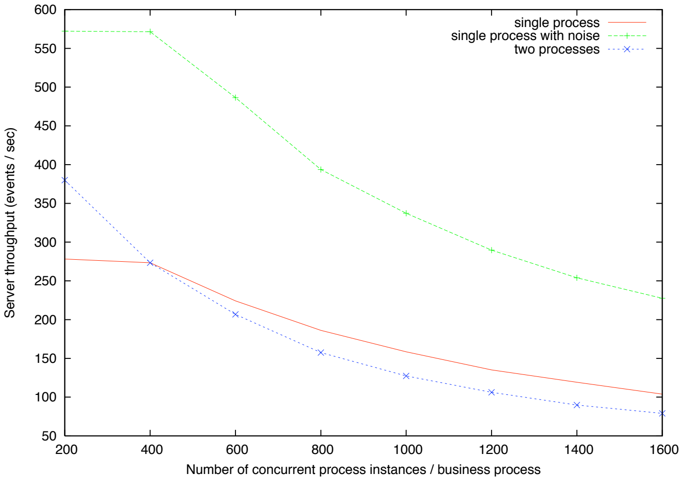


Figure 8. Event processing server's performance

In the single-process scenario, there seems to be an almost constant throughput until 400 concurrent process instances are run. Beyond that, the throughput declines albeit gradually. The single-process-with-noise scenario results in an almost identical plot as the first scenario, just a scale higher on the graph. This tells us that the server throughput is not degraded by the noise and the increase to a higher scale accounts for noise events. The two-process scenario suggests an acceptable scaling of this approach. In this scenario, whereas twice the number of concurrent process instances are running as compared to the single-process scenario, the throughput is not significantly lower. Moreover, for less than 400 concurrent processes instances per business process, the throughput is higher in the two-process scenario.

## VI. Related Work

Compliance assurance can be broadly categorized into two main strategies: "compliance by design", i.e., incorporating compliance directives into a system at design time, and "compliance by detection", i.e., observing a system to ensure that its execution was compliant [14]. We consider a number of related works that are to do with the second approach, i.e. compliance by detection.

Giblin et al. [15], [16] propose REALM, a metamodel for the specification of compliance regulations in a technology-agnostic manner. A REALM model consists of a concept model that captures the concepts and relationships of the domain in which regulations are being applied, a compliance rule set that represents the regulatory requirements in a real-time temporal object logic, and meta-data providing information about the regulations, e.g. source and enactment dates. They also present a framework to that uses the REALM specification to generate technology specific correlation rules for runtime monitoring. In our approach, we assume that compliance controls are already defined within a business process before we identify the different business activities and service invocation event trails that the the process is composed of. However, our approach also drives the creation of rules and queries that are finally used for correlation and monitoring of events.

Vaculín and Sycara [17] propose semantic monitoring of web services also through correlation/aggregation of primitive events into composite events. They achieve this by extending an event algebra to enable specification of composite events. These concepts are very similar to that low-level, service invocation events and the higher level business events, however, their work focuses on monitoring for semantic web services. They do not filter events based on syntax and parameters; rather they define an event ontology and whenever a primitive event is fired, it is actually an instance of the ontology class representing the event type on which semantic filtering is performed.

Policy based runtime monitoring approaches that focus on monitoring of the business process as it executes are proposed by Erradi et al. [18] and Baresi et al. [19]. In both approaches, the monitoring logic is woven within the process control flow. Erradi et al. [18] propose a policy-based approach to runtime monitoring and adaptation of web service compositions. They achieve this through the use of MASC, which is web service middleware that performs monitoring and adaptation. MASC uses monitoring policies specified with WS-Policy4MASC, which is the authors' extension to the WS-Policy Framework. They perform message correlation by logging events / messages to a database and having this database analyzed by the correlation rules to identify desired aggregate business events. Baresi et al. [19] propose WS-CoL (Web Service Constraint Language), a domain independent language that is used to express monitoring policies for WS-BPEL processes. WS-CoL is compliant with the WS-Policy specification. Their approach proposes weaving the monitoring directives into the WS-BPEL specification such that calls to a monitoring manager are attached to certain parts of the specification. This weaving is done at deployment time to keep a separation between the WS-BPEL code and the WS-CoL monitoring constraints. During runtime, whenever these constraints are encountered, the monitoring manager performs constraint checking and then calls the relevant service. In both these approaches the monitoring logic is tightly coupled with the actual running system.

Finally, business protocols [20], [21] are related to our work in the sense of monitoring web services through the interception and observation of messages sent and received by services in order to monitor their execution/interaction patterns. Li et al. [20] propose a framework to monitor and validate the runtime behaviour of web services against pre-defined interaction constraints. Their approach relies on the interception of every message that goes across a service. In their framework interaction constraints are expressed using finite state automata (FSA) and they have a monitoring component that uses these FSAs to validate the web service interactions.

## VII. Discussion

In this section, we discuss how our approach may enable organizations to systematically and sustainably implement compliance in their service-oriented systems.

We propose a stepwise process that moves progressively from business processes to rules and queries for compliance detection. For technical personnel in an organization, there is a separation of issues concerning the system functionality and issues concerning compliance assurance. As a result, maintenance overheads that are incurred without this clear separation are reduced. This separation characteristic is also present in related works from the previous section. However, when it comes to compliance issues, we assume that business specialists have already incorporated their choices on compliance controls into the business process designs, and we make a mapping of these choices to the technical implementation of monitoring logic. Other works like Giblin et al. [15], [16] tackle expressing these compliance controls (as compliance rule sets [16]) and then mapping them to the technical implementation.

Considering that compliance requirements are ever-changing, organizations have to continuously adapt their compliance concerns. Our approach provides a clear change strategy: Whenever a compliant business process is changed, the change impact affects a set of event trails, which in turn affect a set of CEP rules. Hence, the explicit trace links in our approach foster understandability, changeability, and maintainability of our event-based compliance solutions.

The identification of event trails might foster reuse of compliance rules. Whenever another business process can be mapped to the same sequence of technical events, we can identify the same business events. Hence, even if the business process activities are not the same but can be mapped to the same event trail, reuse of existing compliance rules is possible.

We propose generating service invocations events to monitor business processes. These events have a standard format and only change the values of parameters depending on the service invoked. In an organization with many services (as is the case in large scale SOAs), we are able to reuse the event generation code across these services. The approach proposed by Giblin et al. [16] expresses compliance regulations in a technology-agnostic manner and finally generates the technology-specific correlation rules. This improves reuse in situations where the runtime monitoring technology might change.

Finally the compliance detection rules/queries that are used in CEP tools are in most cases written in simple query languages like shown in Listing 1. They are thus relatively easy to understand by technical personnel. Erradi et al. [18] and Baresi et al. [19] use WS-Policy based languages that are expressed in XML – making these languages readable and perhaps even providing opportunities for automated processing and transformations. The other monitoring approaches have more complicated expression languages for the compliance concerns. Giblin et al. [15], [16] use a compliance rule set based on temporal object logic, while Vaculín and Sycara [17] uses an event algebra to aggregate events.

## VIII. Summary and Conclusion

Compliance with regulations, laws and policies is a requirement for organizations to avoid negative consequences. These organizations thus have to monitor their information systems to ensure that they still adhere to these compliance concerns. Considering that many organizations today implement their systems based on process-driven service-oriented architectures, we are proposing an approach for monitoring business processes for compliance in such process-driven SOAs.

We propose a stepwise approach moving from business processes, breaking them down to business activities, and identifying the relevant service invocations needed to realize each business activity. In order to monitor the business activities, we represent each service invocation as an event. The event trails, that is events representing the service invocations of a business activity, drive the creation of queries used to monitor business processes, and detect compliance violations. We evaluate our approach on a simple monitoring infrastructure, and run a number of test scenarios that mimic a large-scale process-driven SOA environment.

## Acknowledgments

This work was supported by funds from the European Commission (contract No. 215175 for the FP7-ICT-2007-1 project COMPAS).

## References

[1] U. Zdun, C. Hentrich, and S. Dustdar, “Modeling process-driven and service-oriented architectures using patterns and pattern primitives,” *ACM Trans. Web*, vol. 1, no. 3, p. 14, 2007.

[2] P. Kung, C. Hagen, M. Rodel, and S. Seifert, “Business process monitoring & measurement in a large bank: challenges and selected approaches,” in *Database and Expert Systems Applications, 2005. Proceedings. Sixteenth International Workshop on*, Aug. 2005, pp. 955–961.

[3] C. McGregor and S. Kumaran, “Business process monitoring using web services in B2B e-commerce,” in *Parallel and Distributed Processing Symposium., Proceedings International, IPDPS 2002, Abstracts and CD-ROM*, 2002, pp. 219–226.

[4] M. zur Muehlen and M. Rosemann, “Workflow-based process monitoring and controlling-technical and organizational issues,” in *System Sciences, 2000. Proceedings of the 33rd Annual Hawaii International Conference on*, 2000, p. 10 pp. vol.2.

[5] J. G. Kang and K. H. Han, “A business activity monitoring system supporting real-time business performance management,” in *Convergence and Hybrid Information Technology, 2008. ICCIT ’08. Third International Conference on*, vol. 1, Nov. 2008, pp. 473–478.

[6] T. Greiner, W. Düster, F. Pouatcha, R. von Ammon, H.-M. Brandl, and D. Guschakowski, “Business activity monitoring of norisbank taking the example of the application easy-credit and the future adoption of complex event processing (CEP),” in *PPPJ ’06: Proceedings of the 4th international symposium on Principles and practice of programming in Java*. New York, NY, USA: ACM, 2006, pp. 237–242.

[7] S. Rozsnyai, R. Vecera, J. Schiefer, and A. Schatten, “Event cloud - searching for correlated business events,” in *E-Commerce Technology and the 4th IEEE International Conference on Enterprise Computing, E-Commerce, and E-Services, 2007. CEC/EEE 2007. The 9th IEEE International Conference on*, July 2007, pp. 409–420.

[8] “National institute of standards and technology special publications 800-30: Risk management guide for information technology systems,” July 2002.

[9] D. C. Luckham, *The Power of Events: An Introduction to Complex Event Processing in Distributed Enterprise Systems*. Reading, MA: Addison-Wesley, 2002.

[10] H.-M. Brandl, “Complex event processing in the context of business activity monitoring,” Master’s thesis, University of Applied Sciences Regensburg, 2007.

[11] E. Wu, Y. Diao, and S. Rizvi, “High-performance complex event processing over streams,” in *SIGMOD ’06: Proceedings of the 2006 ACM SIGMOD international conference on Management of data*. New York, NY, USA: ACM, 2006, pp. 407–418.

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

[13] http://activemq.apache.org.

[14] S. Sackmann, M. Kahmer, M. Gilliot, and L. Lowis, “A classification model for automating compliance,” in *10th IEEE Conf. EEE/CEC*, July 2008, pp. 79–86.

[15] C. Giblin, A. Y. Liu, and X. Zhou, “Regulations expressed as logical models (REALM),” in *Proc. of the 18th Annual Conference on Legal Knowledge and Information Systems (JURIX 2005)*, A. I. O. S. Press, Ed., 2005, pp. 37–48.

[16] C. Giblin, S. Müller, and B. Pfitzmann, “From regulatory policies to event monitoring rules: Towards model-driven compliance automation,” IBM Research, Tech. Rep. RZ 3662, 2006.

[17] R. Vaculin and K. Sycara, “Specifying and monitoring composite events for semantic web services,” in *Web Services, 2007. ECOWS ’07. Fifth European Conference on*, Nov. 2007, pp. 87–96.

[18] A. Erradi, P. Maheshwari, and V. Tosic, “WS-Policy based monitoring of composite web services,” in *Web Services, 2007. ECOWS ’07. Fifth European Conference on*, Nov. 2007, pp. 99–108.

[19] L. Baresi, S. Guinea, and P. Plebani, *WS-Policy for Service Monitoring*, ser. Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 2006, pp. 72–83.

[20] Z. Li, Y. Jin, and J. Han, “A runtime monitoring and validation framework for web service interactions,” in *ASWEC*, 2006, pp. 70–79.

[21] B. Benatallah, F. Casati, and F. Toumani, “Analysis and management of web service protocols,” in *Conceptual Modeling - ER 2004, 23rd International Conference on Conceptual Modeling, Shanghai, China, November 2004, Proceedings*, 2004, pp. 524–541.