Security Policy Monitoring of BPMN-based Service Compositions

Muhammad Asim[a], Artsiom Yautsiukhin[b], Achim D. Brucker[c,*], Thar Baker[d] ,Qi Shi[d], Brett Lempereur[d]

[a]*Department of Computer Science, National University of Computer and Emerging Sciences, Islamabad, Pakistan*
[b]*IIT-CNR, via Moruzzi, 1, Pisa, Italy, 56124*
[c]*Department of Computer Science, The University of Sheffield, Sheffield, UK*
[d]*Department of Computer Science, Liverpool John Moores University, Liverpool, UK*

***Abstract*— Service composition is a key concept of Service-Oriented Architecture that allows for combining loosely coupled services that are offered and operated by different service providers. Such environments are expected to dynamically respond to changes that may occur at runtime, including changes in the environment and individual services themselves. Therefore, it is crucial to monitor these loosely-coupled services throughout their lifetime. In this paper, we present a novel framework for monitoring services at runtime and ensuring that services behave as they have promised. In particular, we focus on monitoring non-functional properties that are specified within an agreed security contract. The novelty of our work is based on the way in which monitoring information can be combined from multiple dynamic services to automate the monitoring of business processes and proactively report compliance violations. The framework enables monitoring of both atomic and composite services and provides a user friendly interface for specifying the monitoring policy. We provide an information service case study using a real composite service to demonstrate how we achieve compliance monitoring. The transformation of security policy into monitoring rules, which is done automatically, makes our framework more flexible and accurate than existing techniques.**

***Index Terms*— Service-Oriented Computing, Composite services, Business process compliance, Compliance monitoring, Security.**

I. INTRODUCTION

Service-Oriented Architecture (SOA) allows software components from different providers to be exported as services for external use. A service itself is a unit that offers a certain functionality. If no single service can satisfy the functionality required by the user, then SOA allows multiple services to be combined to form a larger application to fulfil the user requirements. A SOA platform provides a foundation for modeling and composing multiple services in an "*ad-hoc*" manner. Service descriptions are published by service developers and used by the potential users to discover services. A service composer is a service provider that is responsible for constructing service compositions and offering them to consumers. Service discovery is based on matching user requirements and security needs with the published service descriptions. Typically, service composers will have different needs and different requirements. They have varying business goals and different expectations from a service; for example, in terms of functionality, quality of service and security needs. Thus, it is important to make sure that a service should deliver what it promises and should match the user's expectations. However, SOA-based applications are highly dynamic and liable to change heavily at runtime. These applications are made out of services that are deployed and run independently, and may change unpredictably after deployment. Thus, changes may occur to services after deployment and at runtime, which may lead to a situation where services fail to deliver what has been promised. Traditional verification techniques cannot foresee all of these changes as they are mainly pre-deployment activities. These challenges call for more effective approaches towards runtime monitoring of services [1].[2].[3].[4].[5].[6]..

Service composition can be viewed in a process-oriented perspective. This makes the composition not only easy to understand but also the composition can be validated against the desired rules and modified to suit the required operation. In a process-oriented approach, service composition is described by means of workflow languages and technologies. The workflow composition defines the operations to invoke and the execution order of the invocations [7].. A de-facto standard Business Process Model and Notation (BPMN) [8]. are widely used as a modeling notation for business processes [9]..

This paper focuses on our monitoring framework that is based on the runtime monitoring of a service to ensure that the service behaves in compliance with a predefined security policy. We mainly concentrate on monitoring service behavior throughout the service execution lifetime to ensure that services behave as promised. Alerts regarding policy violations are sent as notifications. Current monitoring methods applied to service execution environments focus on generating alerts for a specific set of pre-built event-types. The dynamic nature of SOAs also extends to the end-user security requirements. An ideal system might allow different users to be awarded the opportunity to apply their own security policies enforced through a combination of design time and runtime checks. This might be the case even where multiple users are accessing the same services simultaneously. The main contribution of our framework is the focus on monitoring composite services and checking their workflow, invoked sub-services, compound properties, etc.

[*] Parts of this research were done while the author was a Security Testing Strategist and Research Expert at SAP SE in Germany.

It also allows different user-specified policies to be monitored simultaneously at runtime with the accuracy of a monitoring system that links directly into the service execution environment. Thus, the framework has the capability not only to reveal the information predefined by the provider, but also to be configured to allow users to specify the monitoring rules (using the properties the service has to comply with). Finally, taking into consideration the limitation of any formal-based

enforcement approach, we provide a possibility to add custom property checks for atomic services. These property checks can be added to our framework without any modification of the formal semantics of the language (ConSpec) and the monitoring service itself. Due to this we can monitor a much wider set of properties allowed by the formal language while still enjoying its advantages.

The preliminary design of the monitoring framework and the event model have been described in a previous work [10].[11].. This paper amends and extends the monitoring framework, addresses its limitations, and evaluates its performance. In the previous work, the monitoring framework was based on the use of Complex Event Processing (CEP). It used one language for requirements specification, ConSpec [12]., and another one, Drools Fusion [13]., for monitoring these requirements. One major limitation of previous work was a missing transformation engine required to translate requirement specifications into monitoring rules. The monitoring rules were defined manually through an external interface. This introduced a lot of complexities, particularly for dynamically changing service compositions, which indicated the need for an automated operation. The work presented in this paper uses one language for the requirement specification and the actual monitoring. This provides a seamless and uniform approach to service monitoring. Our contribution in this paper differs from the existing work, e.g. [10]., as follows:

- A new approach has been proposed where only one language (ConSpec) has been used for both the requirements specification and monitoring rules. The Policy Decision Point (PDP) is developed as a part of the monitoring framework, which helps in translating ConSpec policies into monitoring rules and decision-making.
- The ConSpec policies can be used for expressing temporal properties spawning across several atomic services participating in a service composition. Moreover, they can include meta-properties such as restricting the service provider.
- The monitoring framework is developed as a software module and allows straightforward integration with other modules or platforms.
- We focus on monitoring non-functional properties (i.e., properties related to security and trust) that are specified within an agreed security contract. An information composite service case study (based on real services) has been used to demonstrate the compliance monitoring with a focus on two composite security properties.
- A ConSpec editor has been developed which provides a graphical user interface for making and changing ConSpec policies.
- The paper presents an in-depth performance evaluation to show that our system is well-suited for highly dynamic service compositions, which were missing in the previous work.

The rest of the paper is organized as follows: Section 2 describes a motivating service composition example. A discussion of the policy language is presented in Section 3. Section 4 describes the event model we propose for the monitoring framework. Our proposed monitoring framework is explained in Section 5. Sec. 6 describes the implementation of the proposed monitoring framework. Section 7 describes the assessment of the monitoring framework using a case study. Section 8 presents an in-depth performance evaluation to show that our system is well-suited for highly dynamic service compositions. Section 9 compares our approach with existing work and Section 10 concludes the paper and indicates the direction of our future work.

II. Secure Service Composition: An Example

We will illustrate our approach by using a running example. Fig. 1 presents an overview of the InfoService case study. In this example, we assume a small company that designs, develops, and provides customized services to customers. We also assume that customers want to have an application that provides a location-based information service, e.g., based on the current GPS coordinates of a mobile device or after entering an address. The application should display information such as the current weather or a map highlighting various Points of Interest (PoI).

As there are many services available that already provide such information, it is a quite natural approach to building this new application based on already existing services, e.g.:

1. A GeoCoding type service, which takes a street address as input and produces the associated geographical coordinates;
2. A PointOfInterest type service that takes geographical coordinates (output of GeoCoding service) as input and returns the places that the end user can be interested in;
3. A WeatherForecast type service that takes as input the geographical coordinates and returns the information about the weather predictions at the closest location to the end user;
4. A Map type service that takes potential places of interests originated by (2) as input and returns a map showing the position and distance of the end user to each of these places;
5. A WebPageInfoCollector type service takes a set of information related to a location gained from (3) and (4) as input, and returns a Web page that shows it.

The resulting composite service is named InfoService. Each service in the InfoService composition is bound to a real Web Service running in the background and registered with a Marketplace (e.g., Aniketos Marketplace [14].). After providing the street address of the user as input, the composite service returns a Web page with some information related to the user's location.

Operating even such a simple service composition raises already a number of security (e.g., data privacy, access control, see Brucker, et al. [15]. for a more detailed discussion), trustworthiness (e.g., customers may trust different service parties of a service composition to a different extend, see Elshaafi et al. [16]. for a more detailed discussion), reliability (i.e., services should deliver correct results), and availability (e.g., services should always be available) concerns.

In our example, customers usually consider revealing their current locations as a privacy violation. Thus, the GPS coordinates should only be transferred to the services that actually require such information for their operation. Moving one step further, we see that the WeatherForecast requires an approximation of the location (e.g., the city), while the

PointOfInterest service and Map service need the precise location for producing more accurate and precise results. Secondly, the service provider might need a separation of duty of the PointOfInterest service and WeatherForecast service to prevent fraud. In other words, a malicious provider offering both the PointOfInterest and WeatherForecast services could deliberately predict bad weather conditions for certain areas (i.e., to harm it financially); hence, convinces users to visit different places. Consequently, this threatens the repudiation of the composed WebPageInfoCollector.

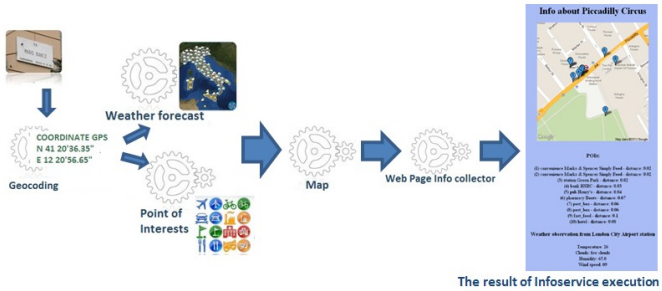

Fig. 1: Overview of InfoService Components.

These requirements need to be considered during the entire lifecycle of the service composition (i.e., from the requirements engineering, to the development, to the operation of the application). Service composition based applications are highly dynamic. Thus, specification and runtime enforcement of security properties is not sufficient [17].. The security, trustworthiness, reliability, and availability of atomic services as well as composed services need to be monitored constantly and, depending on the observations, necessary actions need to be taken. For example, if the monitoring shows evidence that the trustworthiness of a service falls below a certain threshold, a dynamic re-composition should replace this service with another service of a similar kind that satisfies the required trustworthiness level.

III. Policy Language

We need a suitable policy language to specify what we need to monitor. In general, this language should serve for other purposes as well, e.g., it should specify the security requirements for a service (either desired by a consumer or advertised by a service provider). Naturally, we may use one language for requirements specification and another one for monitoring these requirements (as it is done in Asim et al. [10].). In this case, there is a need for a transformation engine between these languages. Thus, one language for both purposes significantly reduces the complexity [18]..

We were looking for a language which could:

- Express security properties and policies for hierarchical services;
- Be expressive enough, clear and simple in processing at the same time;
- Be generated by both humans and software;
- Be able to express complex (security and privacy) policies;
- Be used for requirements specification, matching, monitoring and reasoning.

We considered several candidates, that are exploited by current state of the art frameworks such as WS02 (https://www.ws02.com), for such kind of language. XACML [19]. is a general-purpose policy language, but it is deemed cruel to write policies with it and to reason about them. Moreover, we will need to use the constraint part of policies in a nonstandard way. Event Calculus [20]. is suitable for runtime monitoring and representing policies in a dynamic environment. On the other hand, the syntax of the language becomes too complex for compound services. Furthermore, considerable effort is required to automate generation and runtime monitoring of such policies. The PROTUNE [21]. language has high expressivity and can be used to specify complex policies in a distributed environment. The main disadvantages of the method relates to its strength. Because of such enormous expressivity the language is complex for policy writing and reasoning.

Based on the above analysis, we select the ConSpec language for our purposes. The ConSpec language has been proposed by the University of Trento (UNITN) and Royal Institute of Technology (KTH) in the scope of the Security of Software and Services for Mobile Systems (S3MS) project [22].. Briefly, we can see the language as follows[1]:

The tag RULE ID defines the id of the policy. The tag SCOPE specifies whether the rule is applied to one specific execution or to all executions of the service. The tag SECURITY STATE defines the global variables and their initial values. Then, several events are checked BEFORE or AFTER occurrence.

If an event occurred, we check guards one by one until we find the one that is satisfactory. In this case, certain security updates are performed. If no guards are fired for the event, then the further execution is not permitted (and some further security actions, like notifying the customer, are triggered). In case no security updates are needed but the further execution is allowed, there is a special action “skip”, which does not do anything but continues the execution. There is also a possibility of specifying an ELSE statement for the cases, when the further execution should be allowed even if no guards are fired[2].

```
RULE ID ruleID
   SCOPE <Session | Multisession>
   SECURITY STATE
      <bool | int | string> VarName_1 = <Value_1>
                                ⋮
      <bool | int | string> VarName_n = <Value_n>

   <BEFORE | AFTER> event_1 PERFORM
      Guard_1,1 → Update_1,1
                ⋮
      Guard_1,m → Update_1,m

     ⋮

   <BEFORE | AFTER> event_i PERFORM
      Guard_i,1 → Update_i,1
                ⋮
      Guard_i,j → Update_i,j
```

Fig. 2: The Concrete Syntax of ConSpec.

1 We refer the reader to Aktug and Naliuka [8] for more details, and Fig. 2

2 We omitted this option here for simplicity

The ConSpec language can be straightforwardly mapped to the ConSpec automata. This automata can be seen as $A = (Q, T, \delta, q_0)$, where Q is a set of states and $q_0 \in Q$ is an initial state, T is a set of actions, and δ is a (partial) transition function $\delta : Q \times T \rightarrow Q$. A state can be seen simply as a specific assignment to the variables defined in the SECURITY STATE part. Naturally, the assignment defined in the SECURITY STATE part defines q_0. Actions are defined by the guarded events (specified between BEFORE, AFTER, and PERFORM), i.e., by the name of the event (class and method), the set of its parameters and possible assignments for these parameters (in the AFTER case also the results of the event are considered). Finally, the (partial) transition functions join states with the parameters which fire some of the specified guards and the states which are received after the application of the corresponding updates.

There are a number of advantages of ConSpec. First, this language was developed for security purposes and allows guarding possible actions performed by a system. It represents behavior in terms of different events that allow policies to be checked at runtime. Policy written in ConSpec has a comparatively simple semantics, and is simple to learn.

ConSpec is an automata-based language. Although this feature slightly reduces its expressiveness (in comparison with its predecessor PSLan Erlingsson [23]., or other declarative languages as EventCalculus [20]., XACML [19]., PROTUNE [21]., etc.), however, this feature allows automatic reasoning on it. Thus, ConSpec permits defining *complex* policies, which are necessary to specify security requirements, and provides an efficient way to check them. In other words, next to simple value checking policies (e.g., verifying that the trustworthiness level of a service is higher than some threshold), ConSpec is devised to monitor complex logical constructions (i.e., security policies) without the need for an additional layer of logical verification. In addition, ConSpec provides the embedded facility to evaluate properties before/after the monitored events, which is required particularly by Application security to prevent potentially malicious events from happening. Furthermore, the language is straightforward to define a policy decision point for monitoring purposes if an automaton is available. Also, ConSpec defines different scopes of its application. Thus, we may define a policy for a single execution of a service or multiple executions.

Being based on basic programming and logical rules, ConSpec is an easy to learn language. Nevertheless, for most common applications, the details could be even hidden from the policy maker: an expert may define a template for a policy (i.e., a ConSpec Rule, where only initial Security State should be instantiated) and users will be required to simply provide the required input. This capability is very important for services, where the same security policy could be applied to any service for some hierarchical piece of business process: the only thing to be done is to instantiate the same selected template for low level services.

IV. Event Model

The monitoring framework we propose is built around the concept of events. It is an event-driven approach that allows the monitoring system to analyze events and react to certain situations as they occur. Any viable monitoring system must have the ability to analyze and identify the correct events in a timely manner. Fig. 3 displays a simplified version of our proposed event model. This organizes different event types allowing us to reason about and provide a generic way to deal with them.

Event Listeners are embedded into the BPMN specifications which are triggered by events during workflow execution. These listeners can be configured at the Process level, Activity level or Transition level to generate events.

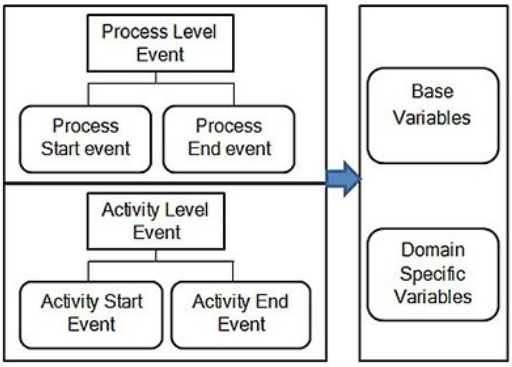

Fig. 3: Event Model.

Our event model is based on two types of process variables; Base Variables and Domain Specific Variables. Both types of variable are available during the execution of a business process and could be used for monitoring. The listeners have access to these process variables and can create events populated using their associated values, sending for analysis. The Base Variables inherit common attributes from the process itself, e.g., the process ID, process name, activity ID, activity name, process start time, etc. However, the Domain Specific Variables, declared in the SECURITY STATE section of ConSpec rules, are user defined and may build upon the Base Variables. For example, to analyse the load on a particular service, we could accumulate all start processing events for that service over the last hour. An alert message should be generated if the number of requests is more than a threshold value in the last hour. This threshold value is a user-defined attribute falling within the Domain Specific Variables.

In the following discussion, we try to determine the structure of events that should be received for analysis. In our proposed framework, an overall process represents a composite service and an activity represents a service component. Fig. 4 shows an example of events for the InfoService BPMN process executed in a specific order.

In this example, the InfoService BPMN process comprises five service tasks each with a *Start* and *End* event. The monitoring of an activity may need only the process ID, activity start and end events. The selection of start and end events for listening is determined by the nature of the monitor. With such events we can aggregate the data received so far and analyze it before (after) execution of a specific activity. Thus, we can prevent invocation of a service if it potentially can violate the contract. Then, the alarm rule is fired and appropriate reaction may be carried out, e.g., change the service fulfilling the activity.

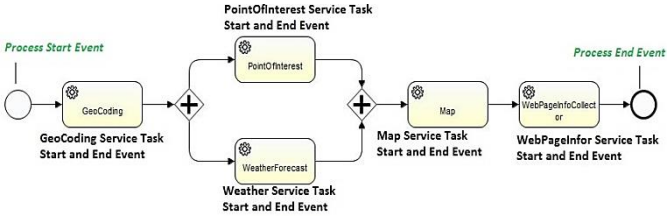

Fig. 4: Event Flow.

In our proposal, an event structure describes the data and structure associated with an event. It helps in organizing the data that is required for monitoring. Below we define the event structure for our proposed monitoring framework.

1. Process level event
   processName
   eventLevel (processLevelEvent)
   eventName (Start or End)
   eventTime (Timestamp)
   Variable 0 . . . n – domain specific variables
2. Activity level event
   processName
   activityName (name of the Service or User Task)
   eventLevel (activityLevelEvent)
   eventType (Service Task or User Task)
   eventName (Start or End)
   processFlow (used to construct a composition work-flow)
   eventTime (Timestamp)
   Variable 0 . . . n – domain specific variables
   eventDate (e.g., 2013/04/05)

V. The Security Policy Monitoring Framework

Our monitoring framework is a software module that runs in parallel to a BPMN process and observes its behavior by intercepting the events that are produced by the processes. The framework we describe is modular and allows a simple integration with other modules or platforms. The components of the monitoring framework are illustrated in Fig. 5 and a description of each of the monitoring units follows.

A. Monitoring Policy

A monitoring policy is a set of requirements, defined using the language ConSpec, which specifies what to monitor for a particular BPMN process. These requirements can be specified by a service provider as well as by a service consumer (depending on the contract specification process).

B. Monitoring Rule Repository

It is a database of monitoring rules used for monitoring services. The rules defined in the monitoring policy are translated into monitoring rules by the monitoring module and are stored in the Monitoring rule repository. An example of a monitoring rule might specify that the trust value of a service should be continuously monitored so that a notification is generated as soon as the value falls below a given threshold.

C. Event Manager

This module gathers the events coming from the runtime environment (running the BPMN processes) and passes them to the Analyzer. The event manager is composed of an Event Filter that filters relevant events for compliance monitoring. The Event Filter relies on a filtering mechanism and acts as a first step to reduce the number of events that must be considered by the Analyzer.

D. Analyzer

Upon receiving events from the Event Manager, the Analyzer analyses them by accessing rules from the repository. It uses the monitoring policy to select the appropriate monitoring rules for a particular process.

Every policy is analyzed according to the ConSpec specification. In particular, if a policy has a Scope "Session" the policy is initialized when a service is invoked. For "Multi-Session" policies the initialization is performed when the service is added/registered in the platform. On the level of ConSpec initialization, it means that we start with the q0 state, assigning initial values specified in the SECURITY STATE part of the rule to the set of declared parameters.

The Policy Decision Point (PDP) is developed as a part of the Analyzer. The PDP helps in translating ConSpec policies into monitoring rules and decision making. It also uploads the initial values of global variables (i.e., specified in the Security State part of the policy) to the memory.

Upon receiving events from the Analyzer, the PDP analyses them according to the order of the guard-update statements specified in the policy. The first guard returning "true" fires the corresponding update (i.e., actions, which have to be performed before continuing of the execution) and afterwards no more statements are checked. Thus, no conflicts are allowed to occur. Firing "true" means that we move from a previous state of the ConSpec automata to the state with the updated values of the considered parameters.

If no guards resulted to "true" (and no updates for ELSE are specified), this means a violation of the policy, i.e., we try to make a transition which does not belong to the automata. If no updates are necessary for some conditions, a special command "skip" is envisaged. For example, a user might specify a policy (Fig. 16 in the Appendix section) to monitor the Map service for trustworthiness every time it is invoked. As BEFORE statement states, the property is updated before an activity starts and if the current activity is anyone but the one we would like to check the trustworthiness value of ("Payment" in the example) or the value is greater than the defined threshold ("90" in the example) nothing happens (skip command). The alarm is raised otherwise (i.e., when the "Payment" service has trustworthiness less than 90). Note that the property requires a special external function (i#TrustworthinessPrediction) to be invoked, which is supported by our framework subject to a proper declaration of the function in the initialization file prior to invocation.

When the ConSpec policy is received by the monitoring module, the Analyzer stores the policy as rules in the repository (memory). When input (event) arrives, the PDP is invoked to change its state and make a decision based on the rules stored in the memory. In this example, if the event

corresponds to the invocation of the Map Service, the PDP will be invoked to retrieve the trustworthiness value and check it against the threshold stored in the memory. If the current trustworthiness value falls below the threshold, a notification will be generated.

The notification alerts are generally in the following format:

alert("ServiceID", Type, Property); Where

ServiceID= “ID of the service involved”

Type=”the type of the notification i.e. Contract Violation”

Property=”the security property agreed to be monitored in the agreed policy but the service failed to adhere i.e. separation of duty”.

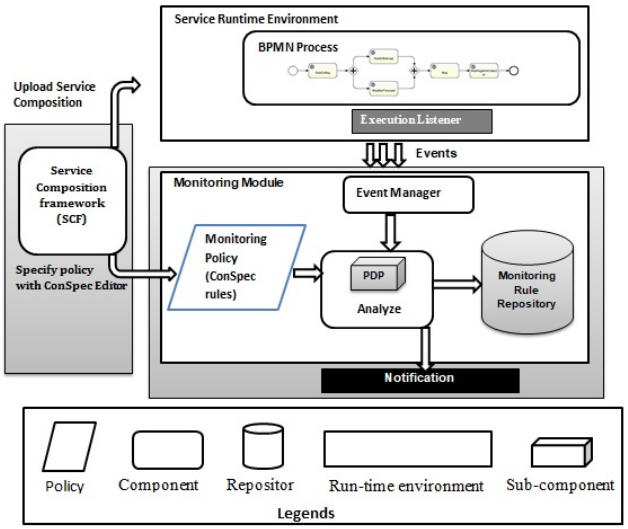

Fig. 5: Security Policy Monitoring Framework

VI. Implementation

We have implemented the monitoring module as an independent service that can easily be integrated into different service frameworks. As an example, we integrated the monitoring module into the Aniketos platform [14]., where it is a part of the Security Monitoring & Notification package. The modules are running in Karaf [24]. as remote OSGI services. Thus, our monitoring framework is not just a part of the overall implementation of the project, but a stand-alone package, which can be easily used in different service composition frameworks such as [25]. or [26]., as long as its interfaces are respected. The only dependency on the Aniketos platform, which the monitoring package has, is verification of precise, non-pure ConSpec, properties (e.g., trustworthiness). On the other hand, our package contains a mechanism, which allows for development of a custom verification module and binding it with the monitoring capabilities. Thus, if one does not want to use the standard Aniketos trustworthiness module but still wants monitoring trustworthiness properties, he/she can use an alternative implementation with a properly defined interface. The source code for the monitoring module is available at GitHub [27]..

As it has been discussed in Section V, the monitoring module (or Security Monitoring & Notification package, as a whole) requires ConSpec policies and the event of the running service as input and produces notifications as output. Next, we briefly describe the modules producing the input for a complete description of the implementation tested in this paper.

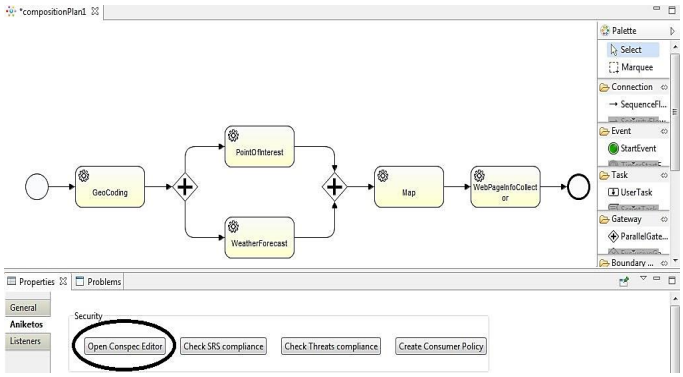

Fig. 6: Service Composition Framework (SCF).

In our implementation ConSpec policies are produced by the Service Composition Framework (SCF) using the integrated ConSpec editor. In general, the SCF (shown in Fig. 6) is an Eclipse-based environment which enables service designers to build executable composition plans and specify their monitoring policies using the ConSpec editor. It allows the modelling of service compositions in BPMN and their deployment to the process execution engine (Activiti engine [24]). The ConSpec properties can be specified by a service provider or a service consumer (depending on the contract specification process performed by SCF).

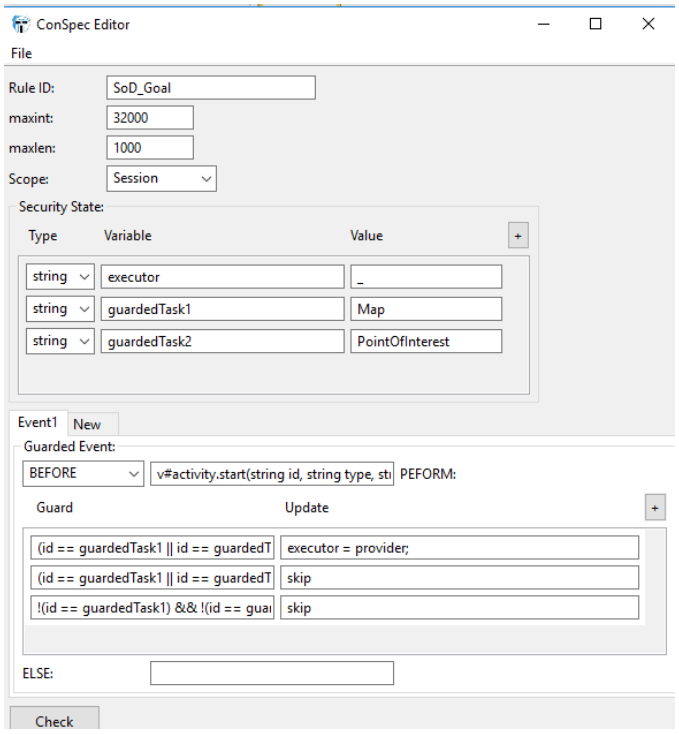

Fig. 7: ConSpec Editor.

We have created a ConSpec editor which provides a graphical user interface for making and changing ConSpec

policies (Fig. 7). The tool also converts the policy to a specified XML format, which simplifies policy processing by the policy decision point (PDP) of the monitor (see Sec. 5). The tool checks the correctness of the written policy and notifies the writer about possible errors.

A Service Runtime Environment (SRE) is responsible for the execution of services (with Activiti engine) and enforcing rules specified by the service designer. SRE interacts with the monitoring module and generates events for the running service that are then analyzed by the monitoring module in compliance with a contract (ConSpec rules).

Finally, SRE is also responsible for handling the notifications generated by the monitoring module. A set of rules can be defined to handle these alarms. For each rule, the service designer can specify the constraints for the event to fire the rule and the action to be performed once the rule is fired. For example, an action could be a recomposition, from simply replacing a single service with another one performing the same task. The result of this action will be a different runnable composition plan satisfying the same security requirements as the substituted Web service. Fig. 8 shows the dialogue available into the Service Composition Framework to allow the definition of rules used by the SRE to manage the behavior of the composite service at runtime. For example, if the separation of duty requirement for both PointOfInterest service and the WeatherForecast service is violated, the specified rule will cause a re-composition by replacing the WeatherForcast service with another functionally similar service offered by a different provider. For more details about SRE and the Aniketos platform, in general, we refer the interested reader elsewhere [29]..

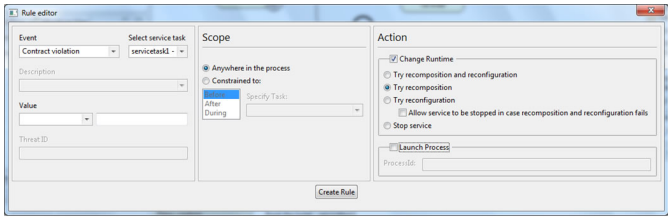

Fig. 8: Rule Editor.

VII. Demonstration Through InfoService

In this section, we demonstrate our monitoring framework using the InfoService example (recall Sec. 2) that has been developed on the Aniketos platform. We consider the separation of duty and binding of duty security requirements for the demonstration below.

The way the atomic services are composed for the InfoService (using the Service Composition Framework) is shown in Fig. 6. The service designer specifies the monitoring policy through the ConSpec editor and wants the following compliance and security requirements for the "information service" process:

1. **Separation of duty**: Both the PointOfInterest service and the WeatherForecast service should be offered by different service providers.
2. **Binding of duty**: The Map service and the WeatherForcast service should belong to the same service provider.

Before service deployment, it is setup that if at runtime the security requirements are not fulfilled then the composition has to be recomposed (with an editor shown in Fig. 6). Finally the composite service along with its monitoring policy is uploaded to the Service Runtime Environment (SRE).

```
RULE ID SoD_Goal
MAXINT 32000
MAXLEN 1000
SESSION session

SECURITY STATE
string executor = _;
string guardedTask1 = WeatherForecast;
string guardedTask2 = PointOfInterest;

BEFORE v#activity.start(string id, string time,
                int time, int date, string provider)
(id == guardedTask1 || id == guarded Task2)
  && executor == "_" -> executor = provider;
(id == guardedTask1 || id == guarded Task2)
  && !(executor == "_") && !(provider == executor)
  -> skip;
!(id == guardedTask1) && !(id == guarded Task2)
  -> skip;
```

Fig. 9: Security policy for separation of duty.

```
RULE ID BoD_Goal
MAXINT 32000
MAXLEN 1000
SESSION session

SECURITY STATE
string executor = _;
string guardedTask1 = Map;
string guardedTask2 = WeatherForecast;

BEFORE v#activity.start(string id, string time,
                int time, int date, string provider)
(id == guardedTask1 || id == guarded Task2)
  && executor == "_" -> executor = provider
(id == guardedTask1 || id == guarded Task2)
  && (executor == "_") && !(provider == executor)
  -> skip;
!(id == guardedTask1) && !(id == guarded Task2)
  -> skip;
```

Fig. 10: Security policy for binding of duty.

After service deployment, the runtime environment forwards the monitoring policy to the monitoring module along with information about the service to monitor, i.e., Service ID. The monitoring policy is a zipped set of XML files, each of which contains one security policy written with the ConSpec language. In other words, a security policy is specified as ConSpec rules in the form of an XML file. These policies state the properties guaranteed by the composite service specification and stated as rules. Each security policy corresponds to a specific security property. For example, Fig. 9 and Fig. 10 show two security policies for both the separation of duty and the binding of duty. Next to these two basic and widely used security properties, a custom policy could be devised with ConSpec. For instance, Fig. 11, shows a custom

policy that states that a user cannot invoke the same PointOfInterest service more than two times (rotating the providers should provide better privacy protection for the user).

```
RULE ID Custome_Privacy
MAXINT 32000
MAXLEN 1000
SESSION multisession


SECURITY STATE
string execRec=_;
string gaurdTask=PointOfInterest;
int count=0;


BEFORE v#activity.start (string id, string
type, int time, int date, string provider)
id==gaurdTask && execRec == provider &&
count < 2
->count=count+1;
id==gaurdTask && !(execRec==provider)
->execRec = provider; count=0;
!(id==gaurdTask)
->skip;
```

Fig. 11: Custom policy example.

During the service execution, five Web Services are executed and events are compiled based on the event model discussed in Sec. 4. The events are then passed to the remote monitoring module for analysis as shown in Fig. 12. The Event Manager relies on a filtering mechanism and acts as a first step to reduce the number of events to be taken into account. Indeed, only events that are considered as relevant for a particular service which needs to be monitored are selected by the Event Manager. This selection has to be carried out according to a particular type of event, the existence of an attribute in an event or a particular value of an event attribute.

```
-------Event received------
processInstanceID=compositionPlan1, eventType=serviceTaskEvent,
eventName=start,eventTime=1388750666591,serviceType=GeoCode,
service Id=GeoCoding, serviceProvider=David

 -------Event received------
processInstanceID=compositionPlan1, eventType=serviceTaskEvent,
eventName=start,eventTime=1388750667160,serviceType=PointOfInte
rest, service Id=PointOfInterest,serviceProvider=Asim

 -------Event received------
processInstanceID=compositionPlan1, eventType=serviceTaskEvent,
eventName=start,eventTime=1388750669075,serviceType=WeatherFore
cast, service Id=WeatherForecast, serviceProvider=Bo
```

Fig. 12: The events received by the monitoring module.

The events are analyzed with the help of the ConSpec PDP (discussed in Sec. 5). The monitoring module analyses these events and triggers alerts to the Notification module in case of any policy violation. In our InfoService case study, both the PointOfInterest service and the WeatherForecast service are offered by the same service provider (violating the separation of duty requirement) and the Map service and the WeatherForcast service are provided by different service providers (violating the binding of duty requirement). This is done deliberately to check if the monitoring framework detects the violation of these security requirements.

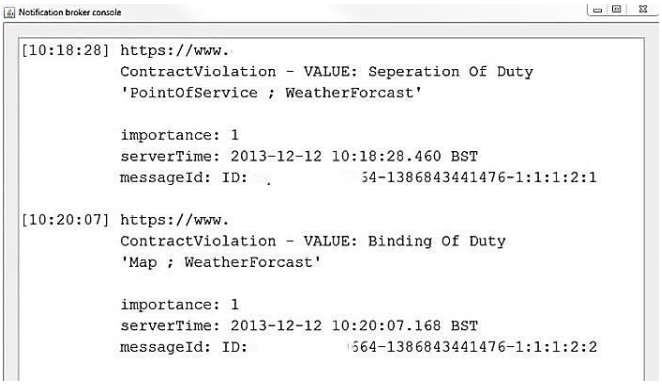

Fig. 13: Notification broker console.

While executing the InfoService process, the monitoring module successfully detected the violation of both the security requirements as shown in Fig. 13. The Notification broker console is developed as a part of the Notification module to monitor the alerts sent to the Notification module. According to the rule set at design time (in case of Separation of duty requirement), a recomposition is triggered and leads to the substitution of the WeatherForcast service with another WeatherForcast service offer by the provider other than the one who provides the PointOfInterest service.

VIII. EVALUATION

This section intends to evaluate how the monitoring module of the proposed framework behaves under high load and to pragmatically infer the number of services that can be monitored by one instance. The performance and scalability of the monitoring framework is mainly influenced by two factors: the complexity of the policy and the number of services being monitored. Our framework can easily scale horizontally: we can easily add as many instances of the monitoring module to ensure that each monitoring module needs only to monitor a "reasonable" number of services. Back to the main point, to evaluate how a single monitoring module behaves under high load and to practically infer how many services can be monitored by one instance, we used the The Grinder[3], a Java-based load testing distributed framework. We implemented a script for the load-testing platform that generated random service identifiers, loaded a set of security policies, and sent a pre-planned script of events to the compliance monitor. This was executed by eight threads for a period of approximately 70-minutes. The logs produced during this test were then processed using The Grinder Analyzer[4].

3 http://grinder.sourceforge.net/
4 http://track.sourceforge.net/

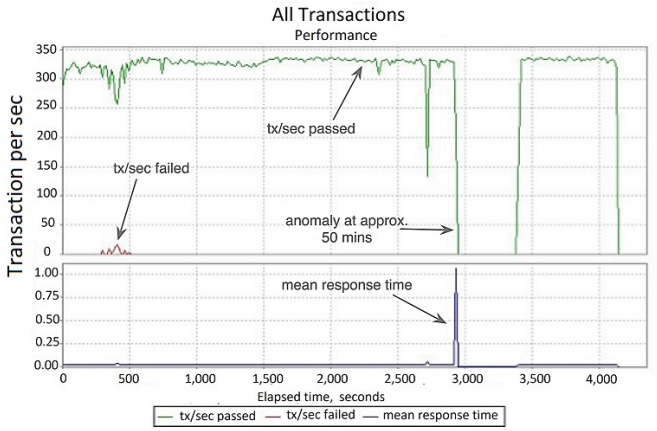

Fig. 14: Graph of transactions per-second (upper plot) and mean response times (lower plot) for the compliance monitor implementation.

Fig. 14 shows the number of transactions processed per second by the compliance monitor whilst under load from eight clients. The number of transactions processed reflects (or at least gives a rough indication on) the complexity of the monitoring policy – the more transactions, the more complex policy, and vice versa. This graph shows that the compliance monitor is capable of handling heavy loads, responding for the majority of the tests between 300-340 transactions per second with response times that were generally under one-second. The anomaly that occurs approximately fifty-minutes into the test, shown in Fig. 14, is caused by periodic garbage collection of many flyweight objects instantiated during the monitoring process and will be corrected for later releases.

Table 1: Load-testing results

Operation	Passed Tests	Failed Tests	Pass Rate	Response Time (seconds)	
				Mean	SD
Create Monitor	134,287	0	100.0%	0.71	2.95
Destroy Monitor	133,295	0	100.0%	0.43	1.1
Process Event	936,432	984	99.9%	34.93	1,344.1
Total	1,204,014	984	99.9%	27.3	1,185.45

Table. 1 provides a breakdown of the performance information by operation. The Web service interface of the compliance monitor implementation supports three operations: creating a monitor for a service and a set of security policies, destroying a monitor when it is no longer required, and processing an event generated by a service to verify its compliance to the policy.

The operations to create and destroy monitors were invoked approximately 134,000 times during the load test, with the discrepancy between creating and destroying monitors explained by a combination of the 984 failed process event operations and the manual termination of the load-test after approximately 70-minutes. Significantly more invocations were made to the process event operation, which in the overall majority of cases responded in less than one-second. This also includes a large number of events which were not in compliance with the monitoring policy in place. These events were ignored and did not cause any notifications to be generated.

Out of the 984 failed tests, 500 customized tests were conducted where the processing event attributes were not in the correct order, or the values were incorrect. Thus, the monitoring module was not in a position to process them and resulted in failed process event operations. The remaining 484 of the 984 failed tests were caused by the testing tool as it failed to process some events.

To conclude, a single instance of the monitoring module is capable of monitoring many services in compliance with a pre-defined security policy, and performs well even when the monitoring instance is under heavy load. The framework supports a rich collection of events and attributes that apply at the level of services within a service composition.

IX. Related Work

The business operations of today's enterprises are heavily influenced by the Business Process modeling of both internal and external business events. Data collected during the execution of business processes are used for identifying the key performance indicators (KPI) that enable the continuous monitoring and tracking of the process behavior and guarantee its correct execution. A number of approaches exist in the literature that focuses on how KPIs are modeled and transferred into events by a model-driven approach [30].[31].[32].[33].. The work of Ly et al. [34]. developed a framework comparing approaches for monitoring business process compliance based on a well-defined set of monitoring functionalities. They emphasize that existing approaches do not provide a solution that combines an expressive language with full compliance.

SALMon [35]. is a generic framework for monitoring the service-based system lifecycle. The framework is platform independent and flexible. It is able to translate a SLA (e.g., written with the WS-Agreement standard) to the specific type of document, called Monitoring Management Document, which is needed in order to configure the monitor. The measurements are provided to the platform with a push or pull method and the values are checked against the constraints specified in the SLA. In contrast to this work, our framework does not require a translation of rules (since monitoring uses the same ConSpec language) and is created to monitor complex policies (which require more complex logic), followed by checking the constraints for the values. Similar to SALMon, our framework is able to add new measurement functions without any modification of the engine, but by a simple declaration of a new measurement.

In another work [36]., the authors incorporated an Agreement Document Analysis (ADA) module into their framework. This module was aimed to provide an explanation to the monitored values. The rules for analysis are expressed as a Constraint Satisfaction Problem (CSP), so the authors require additional mapping, plus, the expressiveness of CSP rules is bound with the languages used for SLA definition. This allows the rules analyzed to be simple checks that the received values are within the defined limits (at least, the authors do not provide any more policies for analysis). The work of Calabro et al. [37]. presents a framework for performance analysis and optimization of a business process expressed in BPMN. It concentrates on generating an event-based monitoring approach that relies on the collection and evaluation of time and cost-based parameters. Chen et al. [38]. proposed a Web service runtime monitoring method based on a probe, which uses aspect-oriented programming (AOP) to realize the monitoring for Web service abnormalities, running time, reliability and availability. The monitoring mechanism

involves capturing the information of services' exceptions, execution time and status events by inserting an AOP monitoring probe in the original the Web service. The exceptions belong to the Java object, captured by the Java virtual machine once occurred in the running of Web service code. Wu et al. [39]. proposed an AOP-based approach for identifying patterns in BPEL processes. They use a stateful aspect extension allowing the definition of behavior patterns that should be identified. If identified, different actions can be triggered. It also permits monitoring certain patterns by using history-based point-cuts. However, monitoring is limited to instances of a BPEL process. The work presented by Baresi et al. [40].; Haiteng and Zhiqing [41].; Wu et al. [39]. is based on how to monitor dynamic service compositions (BPEL processes) with respect to contracts expressed via assertions on services. Assertions are specified with a special-purpose specification language called WSCoL (Web Service Constraint Language), for applying constraints (monitoring rules) on the service execution. In Haiteng and Zhiqing [41]., the authors proposed a solution to the problem of monitoring Web service instances implemented using BPEL. The solution used a Monitoring Broker to access Web service runtime state information and calculates the Quality of Service (QoS) property values. The Monitoring Broker is devised with the support of aspect-oriented programming that separates the business logic of the Web service from its monitoring functionality. Barnawi et al. [42]. presented a pattern-based process to embed compliance monitoring logic within the process definition. The approach targets BPMN based processes to monitor runtime-related aspects such as timing and resource assignment constraints. A compliance expert is needed to visually specify the compliance rules, which are then embedded within the definition of a business process. While there are a number of related techniques, we believe our framework is novel in its ability to monitor both atomic and composite services. It does not require assertions to define what has to be monitored using a proprietary language. Our framework performs compliance monitoring of complex security properties by using a non-intrusive AOP mechanism and has direct access to the service execution environment.

Martín and Pimentel [43]. proposed using security adaptation contracts, which can adapt service orchestration in a secure way. The authors use several specific constructs to express usual security requirements for services (derived from a number of Web service security standards, like WS-Security), check the process for possible violations and have the main focus on adaptation of the orchestration in order to avoid violation of contract terms. Ciancia et al. [44]. extended the work with a richer specification language (CryptoCCS) and provided transformation from BPMN to this language. Instead, in our paper, we focus on monitoring of security properties, define how and where monitoring actions must be performed, and trigger the notification mechanism. Naturally, the adaptation of a service composition is one of the possible further steps, but this is not the main focus of this paper.

The work presented by Alhamazani et al. [45]. discussed several monitoring tools developed for the cloud computing environment. However, these tools have mainly focused on monitoring the low level aspects of resources deployed in the cloud (e.g., memory, CPU, disk) [42].. Our framework considers the monitoring and compliance of the high level and logical security aspects of the cloud computing business process without relying on any external monitoring component where the entire execution environment is mainly controlled by the cloud providers. CloudWatch [46]. is a monitoring service providing comprehensive monitoring for cloud resources and applications run by customers on Amazon Web Services (AWS). CloudWatch can collect and track metrics, collect and monitor log files, set alarms, and react to changes in AWS resources. Thus, Amazon CloudWatch is a useful monitoring solution for Amazon Cloud users; however, the way, in which monitoring data are gathered, collected and analyzed, is not transparent. Further, it is restricted to AWS products. Nagios [47]. is an open source-monitoring framework that allows monitoring of IT infrastructure to ensure proper functioning of systems, applications, and services. It is designed to utilise a list of plug-ins that would be executed to monitor the target system. However, Aceto et al. [48]. suggested that Nagios is not suitable for a rapidly changing dynamic infrastructure (i.e., SOA-based applications are highly dynamic and liable to change heavily at runtime) and is not suitable for as-is adoption in Cloud scenarios. Plugins in Nagios can be easily developed and leverage its flexibility in a way that could monitor virtually any type of network [49].. Our framework could be seamlessly integrated into Nagios as a plugin to perform the security monitoring of BPMN-based services in compliance with a predefined security policy written in ConSpec.

X. Conclusions

We have presented a monitoring framework for SOA-based systems, which is particularly tailored for detecting security, privacy, and trustworthiness violations of service compositions. The monitoring framework ensures that the service behaves in compliance with a predefined security policy. The approach enables monitoring across multiple composite services, and integrates dynamic changes from various subsystems efficiently with high performance. This monitoring framework is, of course, only one building block of a holistic approach for the secure and trustworthy construction and execution of service compositions: while we did not discuss the related details in this paper, the presented monitoring framework was integrated into the Aniketos platform which supports the design-time and runtime aspects of secure and trustworthy service compositions. Nevertheless, it is implemented as a stand-alone package which can be applied in diverse service orchestrating platforms.

Compared with other existing solutions, the monitoring framework presented here offers flexibility as well as applicability in the context of composite services. To achieve this, the platform supports a rich collection of events and attributes that apply at the level of services within a service composition. We demonstrated this using a real composite service invocation monitored against the user specified security policy. Our proposed monitoring framework provides a user friendly interface for service designers to specify their monitoring policies as ConSpec rules. A policy written in ConSpec is easily understandable by humans and the simplicity of the language allows a comparatively simple semantics. This enables the service designer to easily specify the monitoring requirements for their processes and monitor them using the framework. The novelty of our work stems from the way in which monitoring information can be combined from multiple dynamic services to automate the monitoring of business processes and proactively report compliance violations.

Moreover, service users may specify their own properties (using ConSpec) and include them into the contract for monitoring. Generally, the service composition providers can subscribe to different Alerts through the Notification module. Alerts regarding policy violation are sent as notifications to those who subscribed them, enabling verification and decision making.

We see several lines of future work to increase the applicability of our framework, including:

1. To increase the usability, we are investigating high-level notations (e.g., SecureBPMN [14].[15].) for specifying the properties that need to be monitored. On the one hand, our monitoring framework is very flexible. On the other hand, it results in runtime overheads that can be reduced if certain properties can be guaranteed statically (e.g., based on a formal analysis at design-time) and hence excluded from the monitoring at runtime. We will investigate approaches that allow deciding, on a case-by-case basis, if a property of a given service composition should be validated statically or monitored at runtime.

2. The violations of required properties should be not only detected but also reacted (pro-actively) by an execution framework for service compositions to minimize the overall number of violations as well as ensure the availability. Thus, we need to integrate techniques for dynamic service replacements and service re-composition that require explicit user consent or are completely hidden from the end users.

XI. BIBLIOGRAPHY

[1]. Zhou, B and Shi, Q and Yang, P. 2016. A Survey on Quantitative Evaluation of Web Service Security. In: The 15th IEEE International Conference on Trust, Security and Privacy in Computing and Communications, 23 August 2016 - 26 August 2016, Tianjin, China.

[2]. Asim M, Zarzosa S, Shi Q, Zhou B. 2015. A Policy Specification Language for Composite Services. The Fifth International Conference on Advanced Collaborative Networks, Systems and Applications. St. Julians, Malta.

[3]. Ghezzi, C., Guinea, S., 2007. Run-time monitoring in service-oriented architectures. In: Baresi, L., Nitto, E. (Eds.), Test and Analysis of Web Services. Springer Berlin Heidelberg, pp. 237–264.

[4]. Baker, Thar, Michael Mackay, Amjad Shaheed, and Bandar Aldawsari. "Security-oriented cloud platform for soa-based scada." In Cluster, Cloud and Grid Computing (CCGrid), 2015 15th IEEE/ACM International Symposium on, pp. 961-970. IEEE, 2015.

[5]. Karam, Yasir, Thar Baker, and Azzelarabe Taleb-Bendiab. "Security Support for Intention Driven Elastic Cloud Computing." In Computer Modeling and Simulation (EMS), 2012 Sixth UKSim/AMSS European Symposium on, pp. 67-73. IEEE, 2012.

[6]. Al-Sharif, Sultan, Farkhund Iqbal, T. Baker, and A. Khattack. "White-Hat Hacking Framework for Promoting Security Awareness." In New Technologies, Mobility and Security (NTMS), 2016 8th IFIP International Conference on, pp. 1-6. IEEE, 2016.

[7]. Lemos, A.L., Daniel, F. and Benatallah, B., 2016. Web service composition: a survey of techniques and tools. ACM Computing Surveys (CSUR), 48(3), p.33.

[8]. OMG, 2011. Business Process Model and Notation (BPMN) Version 2.0. Available via http://www.omg.org/spec/BPMN/2.0/ on 26/02/2015.

[9]. Skouradaki, M., Roller, D.H., Leymann, F., Ferme, V. and Pautasso, C., 2015, January. On the road to benchmarking BPMN 2.0 workflow engines. In Proceedings of the 6th ACM/SPEC International Conference on Performance Engineering (pp. 301-304). ACM.

[10]. Asim, M., Llewellyn-Jones, D., Lempereur, B., Bo, Z., 2013. Event driven monitoring of composite services. In: Proceedings of the 2013 International Conference on Social Computing (SocialCom). IEEE, Alexandria, pp. 550 – 557.

[11]. Asim, M., Yautsiukhin, A., Brucker, A. D. Security Policy Monitoring of Composite Services. In Secure and Trustworthy Service Composition: The Aniketos Approach. No. 8900 in Lecture Notes in Computer Science: State of the Art Surveys. Springer-Verlag, Heidelberg, 2014; 192-202.

[12]. Aktug, I., Naliuka, K., 2008. Conspec: A formal language for policy specification. Science of Computer Programming 74 (1–2), 2 – 12, special Issue on Security and Trust.

[13]. Drools Fusion: http://drools.jboss.org/drools-fusion.html

[14]. Brucker, A. D., Dalpiaz, F., Giorgini, P., Meland, P. H., Rios, E. (Eds.), 2014. Secure and Trustworthy Service Composition: The Aniketos Approach. No. 8900 in Lecture Notes in Computer Science: State of the Art Surveys. Springer-Verlag, Heidelberg.

[15]. Brucker, A.D., 2013. Integrating security aspects into business process models. it–Information Technology it–Information Technology, 55(6), pp.239-246.

[16]. Elshaafi, H., and Botvich, D. 2016. Optimisation-based collaborative determination of component trustworthiness in service compositions. Security Comm. Networks, 9: 513–527. doi: 10.1002/sec.985.

[17]. Brucker, A. D., Hang, I., Lückemeyer, G., Ruparel, R., 2012. Securebpmn: Modeling and enforcing access control requirements in business processes. In: Proceedings of the 17th ACM Symposium on Access Control Models and Technologies. SACMAT '12. ACM, New York, NY, USA, pp. 123–126.

[18]. Llewellyn-Jones, D and Asim, M., 2013. Requirements for Composite Security Pattern Specification. In: Second International Workshop on Cyberpatterns 2013: Unifying Design Patterns with Security, Attack and Forensic Patterns, Abingdon, UK.

[19]. OASIS, 2013. eXtensible Access Control Markup Language (XACML) Version 3.0. Available via http: //docs.oasis-open.org/xacml/3.0/xacml-3. 0-core-spec-os-en.pdf on 26/02/2015.

[20]. Shanahan, M., 1999. The event calculus explained. In: Wooldridge, M., Veloso, M. (Eds.), Artificial Intelligence Today. Vol. 1600 of Lecture Notes in Computer Science. Springer Berlin Heidelberg, pp. 409–430.

[21]. Bonatti, P. A., Coi, J. L. D., Olmedilla, D., Sauro, L., 2010. PROTUNE: A Rule-based PROvisional TrUst NEgotiation Framework.

[22]. S3MS: Security of software services of mobile systems, http://cordis.europa.eu/project/rcn/78380_en.html

[23]. Erlingsson, U., 2004. The inlined reference monitor approach to security policy enforcement. Ph.D. thesis, Department of Computer Science, Cornell University.

[24]. Apache Karaf: https://karaf.apache.org/

[25]. Lécué, F., Silva, E., Pires, L. F., 2008. A framework for dynamic web services composition. In: Gschwind, T., Pautasso, C. (Eds.), Emerging Web Services Technology, Volume II. Whitestein Series in Software Agent Technologies and Autonomic Computing. Birkhäuser Basel, pp. 59–75.

[26]. Oster, Z. J., Ali, S. A., Santhanam, G. R., Basu, S., Roop, P. S., 2012. A service composition framework based on goal-oriented requirements engineering, model checking, and qualitative preference analysis. In: Liu, C., Ludwig, H., Toumani, F., Yu, Q. (Eds.), Service-Oriented Computing. Vol. 7636 of Lecture Notes in Computer Science. Springer Berlin Heidelberg, pp. 283–297.

[27]. SPMM: https://github.com/AniketosEU/Security-Monitoring-and-Notification/tree/master/aniketos-policymonitoring

[28]. Activiti engine: http://www.activiti.org

[29]. D'Errico, M., Malmignati, F., Andreotti, G. F., 2011. A platform for secure and trustworthy service composition. In: In Proceedings of the The Fifth International Conference on Cloud Computing, GRIDs, and Vir-tualization (Cloud Computing 2014).

[30]. Koetter, F. and Kochanowski, M., 2015. A model-driven approach for event-based business process monitoring. Information Systems and e-Business Management, 13(1), pp.5-36.

[31]. Calabro, A., Lonetti, F. and Marchetti, E., 2015, August. Monitoring of business process execution based on performance indicators. In Software Engineering and Advanced Applications (SEAA), 2015 41st Euromicro Conference on (pp. 255-258). IEEE.

[32]. Krumeich, J., Mehdiyev, N., Werth, D. and Loos, P., 2015, October. Towards an extended metamodel of event-driven process chains to model complex event patterns. In International Conference on Conceptual Modeling (pp. 119-130). Springer International Publishing.

[33]. Krumeich, J., Weis, B., Werth, D. and Loos, P., 2014. Event-driven business process management: where are we now? A comprehensive synthesis and analysis of literature. Business Process Management Journal, 20(4), pp.615-633.

[34]. Ly, L.T., Maggi, F.M., Montali, M., Rinderle-Ma, S. and van der Aalst, W.M., 2015. Compliance monitoring in business processes: Functionalities, application, and tool-support. Information systems, 54, pp.209-234.

[35]. Oriol, M., Franch, X. and Marco, J., 2015. Monitoring the service-based system lifecycle with SALMon. Expert Systems with Applications, 42(19), pp.6507-6521.
[36]. Muller, C., Oriol, M., Franch, X., Marco, J., Resinas, M., Ruiz-Cortes, A. and Rodriguez, M., 2014. Comprehensive explanation of SLA violations at runtime. IEEE Transactions on Services Computing, 7(2), pp.168-183.
[37]. Calabrò, A., Lonetti, F., Marchetti, E. and Spagnolo, G.O., 2016, September. Enhancing Business Process Performance Analysis through Coverage-Based Monitoring. In Quality of Information and Communications Technology (QUATIC), 2016 10th International Conference on the (pp. 35-43). IEEE.
[38]. Chen, L., Xiong, D., Wang, H. and Zou, P., 2016, June. Web service run-time monitoring and visualization analysis based on probe. In Data Science in Cyberspace (DSC), IEEE International Conference on (pp. 446-451). IEEE.
[39]. Wu, G., Wei, J., Huang, T., 2008. Flexible pattern monitoring for wsbpel through stateful aspect extension. In: Proceedings of the 2008 IEEE International Conference on Web Services. ICWS '08. IEEE Computer Society, Washington, DC, USA, pp. 577–584.
[40]. Baresi, L., Guinea, S., Nano, O., Spanoudakis, G., May 2010. Comprehensive monitoring of bpel processes. IEEE Internet Computing 14 (3), 50–57.
[41]. Haiteng, Z., Zhiqing, S., 2011. Runtime monitoring web services implemented in bpel. In: Proceedings of the 2011 International Conference on Uncertainty Reasoning and Knowledge Engineering (URKE). Vol. 1. Bali, pp. 228 – 231.
[42]. Barnawi, A., Awad, A., Elgammal, A., El Shawi, R., Almalaise, A. and Sakr, S., 2015. Runtime self-monitoring approach of business process compliance in cloud environments. Cluster Computing, 18(4), pp.1503-1526.
[43]. Martín, J.A. and Pimentel, E., 2011. Contracts for security adaptation. The Journal of Logic and Algebraic Programming, 80(3-5), pp.154-179.
[44]. Ciancia, V., Martín, J. A., Martinelli, F., Matteucci, I., Petrocchi, M., Pimentel, E., 2014. Automated synthesis and ranking of secure BPMN orchestrators. IJSSE 5 (2), 44–64.
[45]. Alhamazani, K., Ranjan, R., Mitra, K., Rabhi, F., Jayaraman, P.P., Khan, S.U., Guabtni, A. and Bhatnagar, V., 2015. An overview of the commercial cloud monitoring tools: research dimensions, design issues, and state-of-the-art. Computing, 97(4), pp.357-377.
[46]. Amazon CloudWatch. http://aws.amazon.com
[47]. Nagios. http://www.nagios.org
[48]. Aceto, G., Botta, A., De Donato, W. and Pescapè, A., 2013. Cloud monitoring: A survey. Computer Networks, 57(9), pp.2093-2115.
[49]. Rodrigues, G.D.C., Calheiros, R.N. and Guimaraes, V.T., 2016. GL d. Santos, MB de Carvalho, LZ Granville, LMR Tarouco, and R. Buyya, Monitoring of cloud computing environments: Concepts, solutions, trends, and future directions. In Proceedings of the 31st Annual ACM Symposium on Applied Computing, SAC (Vol. 16, pp. 378-383).

I. APPENDIX

No Delegation policy can be devised as is it written in Fig. 15.

```
RULE ID No_delegation
SESSION session

SECURITY STATE
string allowedProvider="Flight Booking Service";
string gaurdTask="Book Flight";


BEFORE v#activity.start (string id, string type,
int time, int date, string provider)

PERFORM
id!=gaurdTask || allowedProvider==provider

 -> skip
```

Figure 15: No Delegation policy.

The same policy may be applied to all services in a hierarchy if the policy is defined with a template. For example, it is easy to see that in the Trustworthiness policy shown in Fig. 16, the two parameters that make it specific are: ServiceID and Value. Thus, it is enough to devise a ConSpec template that simply requires these two inputs. If we require all sub-services to have the same property, we should simply change the ServiceID parameter for all of them. This, however, is just a facilitating procedure for policy making, and it does not affect the monitoring features, since the property is to be defined as a standalone ConSpec rule in the end. We implement the template handling feature within our ConSpec Editor saving the results as a separate xml file. When a user would like to instantiate the policy, he/she are prompted for the required inputs and the editor forms the policy automatically. Moreover, our CSF is able to devise a policy per lower services in the hierarchy automatically, using a similar procedure.

```
RULE ID Trustworthiness
MAXINT 32000
MAXLEN 1000
SESSION session

SECURITY STATE
string ServiceID = Map;
int Value = 90;
/*  assume trustworthiness is
in [0%,....., 100%] */

BEFORE v#activity.start(string id, string type,
int time, int date, string exec)

ServiceID == id &&
i#TrustworthinessPrediction(id) > Value -> skip;
!(ServiceID == id) -> skip;
```

Figure 16: Trustworthiness