Security Policy Monitoring of Composite Services

Muhammad Asim[1], Artsiom Yautsiukhin[2], Achim D. Brucker[3], Brett Lempereur[1], and Qi Shi[1]

[1] School of Computing and Mathematical Sciences, Liverpool John Moores University, UK
{m.asim,b.lempereur,q.shi}@ljmu.ac.uk
[2] Istituto di Informatica e Telematica, Consiglio Nazionale delle Ricerche, Italy
artsiom.yautsiukhin@iit.cnr.it
[3] SAP SE, Vincenz-Priessnitz-Str. 1, 76131 Karlsruhe, Germany
achim.brucker@sap.com

Abstract. One important challenge the Aniketos platform has to address is the effective monitoring of services at runtime to ensure that services behave as promised. A service developer plays the role that is responsible for constructing service compositions and the service provider is responsible for offering them to consumers of the Aniketos platform. Typically, service consumers will have different needs and requirements; they have varying business goals and different expectations from a service, for example in terms of functionality, quality of service and security needs. Given this, it is important to ensure that a service should deliver for which it has been selected and should match the consumer's expectations. If it fails, the system should take appropriate subsequent reactions, e.g., notifications to the service consumer or service designer.

In this chapter, we present the policy-driven monitoring framework which is developed as part of the Aniketos project. The monitoring framework allows different user-specified policies to be monitored simultaneously. The monitoring is performed at the business level, as well as at the implementation level, which allows for checking the policies of composite services as well as atomic ones. The framework sends an alarm in case of policy violation to notify the interested parties and triggers re-composition or re-configuration of the service.

Keywords: monitoring, secure service composition, security policy, complex event processing, SOA, BPMN.

1 Introduction

Applications based on a Service-Oriented Architecture (SOA) 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 for which they have been selected and no longer satisfy user's expectations. Therefore, there is need to shift towards runtime monitoring of services [1].

One important feature of the Aniketos platform is the effective monitoring of services at runtime to ensure that services behave as promised. This paper presents a monitoring framework that is based on the runtime monitoring of a composite service to ensure that the service behaves in compliance with a pre-defined security policy. Alerts regarding policy violations are sent as notifications. BPMN [2] has been used for modelling and specifying composite services, and the Activiti engine [16] as a Business Process Management Platform. BPMN is widely used as a modelling notation for business processes as well as for executing them in a business process engine [3].

Current monitoring methods applied to service execution environments focus on generating alerts for a specific set of pre-built event-types. However, the dynamic nature of SOAs also extends to the end-user security requirements. An ideal system might allow different users to be given the opportunity to apply their own security policies enforced through a combination of design-time and run-time checks. This might be the case even where multiple users are accessing the same services simultaneously. Current monitoring techniques [4, 5, 6, 7] have not been set up with this flexibility in mind.

In this paper we aim to rectify the above weakness of the existing monitoring work by developing a novel policy-driven monitoring framework that allows different user-specified policies to be monitored simultaneously at run-time with the accuracy of a monitoring system that links directly into the service execution environment.

2 Service Composition: An Example

We will illustrate our approach by using a running example. In this example, we assume that we are a small company that designs, develops, and provides customized services to customers. Moreover, we assume that our customer wants 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 interests.

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

- a GeoCoding type service, which takes as input a street address and gets the associated geographical coordinates;
- a PointOfInterest type service that takes as input the geographical coordinates and returns the places that the end user can be interested in;
- an WeatherForecast type service that takes as input the geographical coordinates and returns the information about the weather observations at the station closest to the end user;
- a Map type service that takes as input the geographical coordinates and returns a map showing the position of the end user;
- a WebPageInfoCollector type service that takes as input a set of information related to a location and returns a web page that shows it.

The resulting composite service, named InfoService, takes as input a street address and returns the web page collecting all the information described above. For more details about this scenario and its implementation, we refer the reader elsewhere [17]. Fig. 1 presents an overview of the InfoService case study.

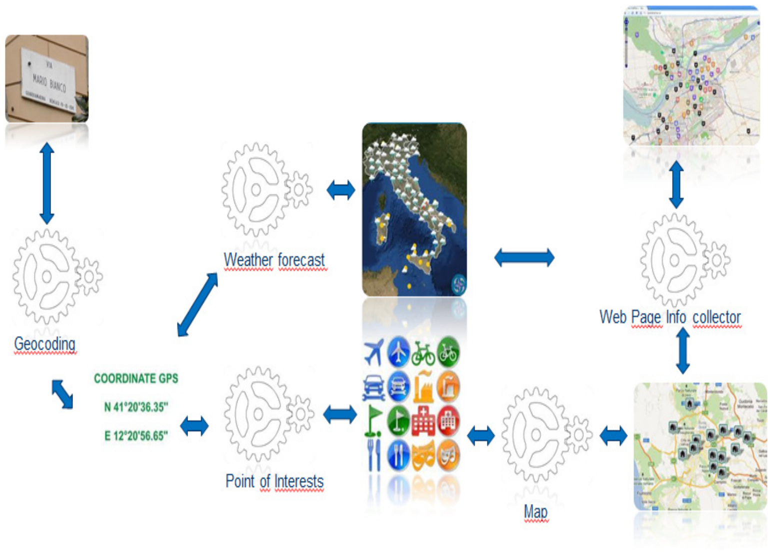


Fig. 1. Overview of InfoService Components

3 Policy Language

In the Aniketos project we were looking for a language which could: (i) express security properties and policies for hierarchical services; (ii) be expressive enough, clear and simple in processing at the same time; (iii) be generated by both humans and software.

We considered several candidates for such kind of language. XACML [9], Event Calculus [10], PROTUNE [11]. XACML is a general purpose language but hard to express policies and reason about them. Event Calculus has a complex syntax for expressing policies for composite services. PROTUNE [17] 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 expressiveness the language is complex for policy writing and reasoning.

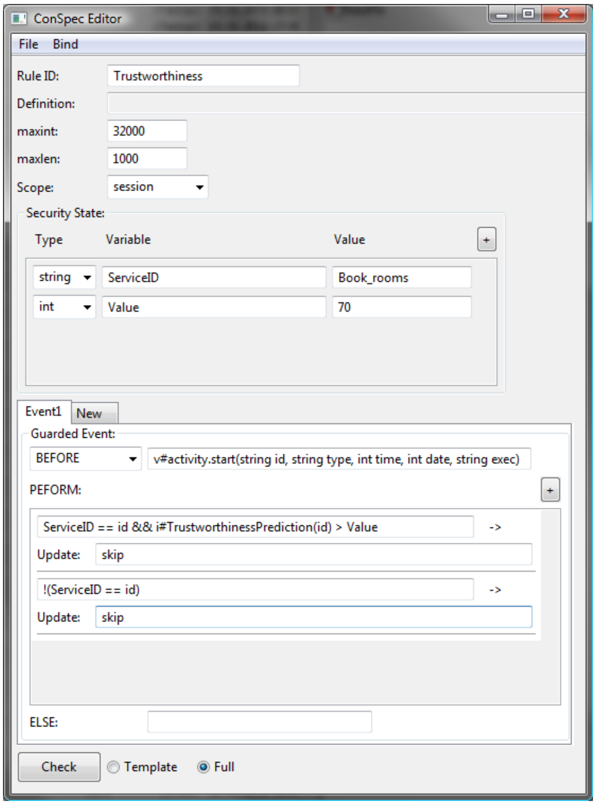
Based on the above analysis, we selected the ConSpec language [12] for our purposes. The ConSpec language was proposed by the University of Trento and Royal Institute of Technology in the scope of the S3MS project [15]. Briefly, we can see the language as follows (we refer a reader to Aktug and Naliuka [12] for the details):

```
RULE ID ruleId
SCOPE <Session | Multisession>
SECURITY STATE
<bool |int|string> VarName1 = <Value1>
<bool |int|string> VarName1 = <Value1>
<BEFORE | AFTER> event1 PERFORM
Gaurd11->Update11
......
Gaurd1N->Update1N
       ...
<BEFORE | AFTER> eventM PERFORM
GaurdM1->UpdateM1
...
GaurdML->UpdateML
```

Fig. 2. ConSpec Syntax

The tag RULEID simply 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 find the one which is satisfied. 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 for specifying an ELSE statement for the cases, when the further execution should be allowed even if no guards fired (we omitted this option here for simplicity).

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 (e.g., a service). It represents behaviour in terms of different events (originally, Java method calls) that allow policies to be checked at runtime. The policies written in ConSpec are easily understandable by humans (the language is similar to programming languages), has comparatively simple semantics, and is easy to learn. ConSpec is an automata-based language. Although this feature slightly reduces its expressiveness (in comparison with its predecessor PSLan [13], or other declarative languages as EventCalculus [10], XACML [9], PROTUNE [11], etc.), it allows automatic reasoning on it. For example, in the project we needed to check that requirements desired by a consumer could be fulfilled by a service provider. Furthermore, it is simple to define a policy decision point for monitoring purposes if automation is available. Finally, ConSpec defines different scopes of its application. Thus, we may define a policy for a single execution of a service or multiple executions.



Rule ID: Trustworthiness
Definition:
maxint: 32000
maxlen: 1000
Scope: session

Security State:

Type	Variable	Value
string	ServiceID	Book_rooms
int	Value	70

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

PEFORM:

ServiceID == id && i#TrustworthinessPrediction(id) > Value ->
Update: skip

!(ServiceID == id) ->
Update: skip

ELSE:

Template [✓ Full]

Fig. 3. ConSpec Editor

In the scope of the Aniketos project we have created a tool which provides a graphical user interface for making and changing ConSpec policies. The tool is called a ConSpec Editor and has been illustrated in Fig. 3. The tool also converts the policy in a specified XML format, which simplifies policy processing by the policy decision point (PDP) of the monitor. The tool checks the correctness of the written policy and notifies the writer about possible errors.

Moreover, the tool allows creating templates for policies, i.e., a predefined policy structure, which requires only initialization of input parameters. Thus, templates significantly simplify the work with ConSpec rules for inexperienced users, who now should simply insert context specific values in a selected policy template. Finally, the tool may be integrated with a service composition framework (e.g., the one shown in Chapters 4 and 9, and retrieve names of used constructs (e.g., IDs of services) or even policies themselves.

4 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 analyse events and react to certain situations as they occur.

Figure 4 displays a simplified version of our proposed event model. This organises different event types allowing us to reason about and provide a generic way to deal with them.

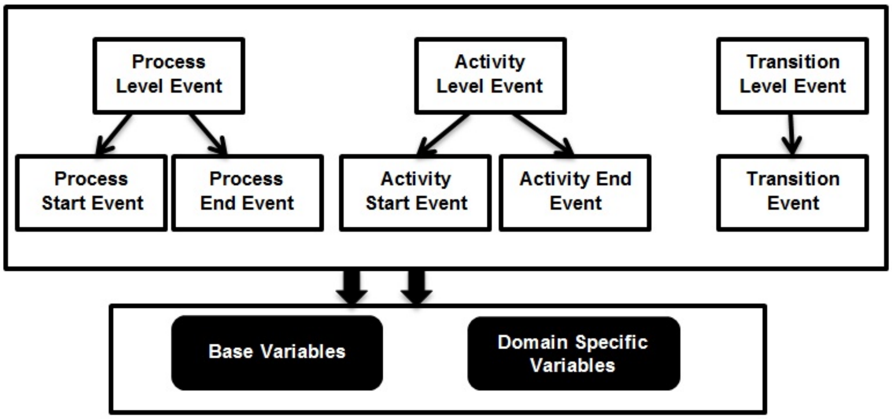


Fig. 4. Event Model

The Activiti engine provides an extension on top of the BPMN 2.0 specification allowing Execution Listeners to be defined. These listeners can be configured at the Process level, Activity level or Transition level in order to generate events. 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. For example, to monitor the execution time of a particular service composition described as a BPMN process (possibly using an extension that supports the specification of security and trust properties [14]), both process start and end events could be used along with the common variables: event start time and event end time. However, the Domain Specific Variables 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 process 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 could represent a composite service and an Activity could represent a service component. Fig. 5 shows an example of events for a BPMN process executed in a specific order.

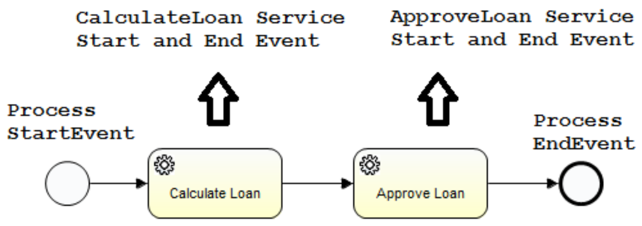

CalculateLoan Service Start and End Event
ApproveLoan Service Start and End Event
Process StartEvent
Process EndEvent
Calculate Loan
Approve Loan

Fig. 5. Event Flow

In this example, a loan service is comprised of loan calculation and loan approval tasks. Therefore, it is not possible to define a single structure for monitoring the overall process. For example, to monitor an Activity, we cannot wait for the whole process to complete. The monitoring of an Activity may need only the process ID, Activity start and end events.

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)

5 The Monitoring Framework

The general architecture of the monitoring framework that we use to monitor the BPMN processes is shown in Fig. 6.

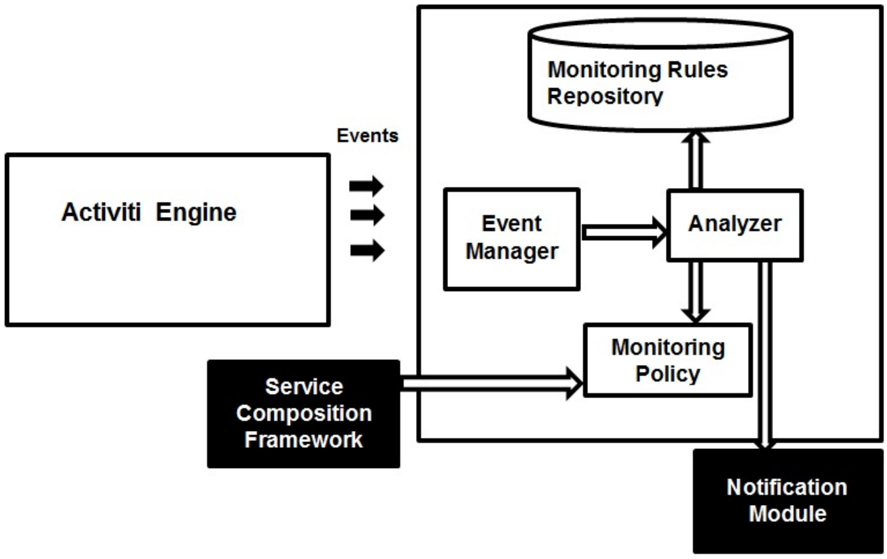


Fig. 6. Monitoring Framework

During execution, the Activiti engine generates events for the deployed BPMN process. The framework consists of an Analyzer that accepts a set of security requirements (monitoring policy) for a particular process to be monitored. The monitoring policy is defined by the service designer. The Analyzer then recovers the monitoring patterns that are related to the requirements from the monitoring pattern repository and checks whether the received events are consistent with the patterns and if it is not then it reports a violation. The monitoring policy is defined using the ConSpec language. The components of the monitoring framework are shown in Fig. 6. In the following, we describe the monitoring components:

Event Manager: This module is responsible for gathering events coming from the Activiti engine and forwards 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.

Monitoring Policy: A set of requirements, specified in ConSpec, that describes what properties need to be monitored for a particular BPMN process. The monitoring policies are defined using the Aniketos Service Composition Framework (SCF), see Chapters 4 and 9.

Consider the following example where a service designer creates a travel booking composition that consists of several tasks, such as ordering, booking hotel, booking flight, payment and invoice, and each task is performed by a component service. The service designer might want that the payment service component should only be invoked when it has a trustworthiness value ≥ 90%. This requirement could easily be specified using the ConSpec language as shown in Fig. 7.

```
MAXINT 32000
MAXLEN 1000
SESSION session

SECURITY STATE
  int trust_threshold = 0.9;
  string ServiceID=PaymentService;


BEFORE v#activity.start(string id, string type,
  string time, string date, string exec)
ServiceID==id   &&   i#Trustworthiness(id)   >
  trust_threshold-> skip;
```

Fig. 7. ConSpec rule for Trustworthiness

Monitoring Rule Repository: It is a database of monitoring patterns used for monitoring services. The rules defined in the monitoring policy are translated into monitoring rules and are stored in the Monitoring Pattern repository. An example monitoring pattern might specify that the trustworthiness of a service should be continuously monitored so that a notification is generated as soon as the value falls below a given threshold.

Analyzer: It analyses the events coming from the Event Manager by using patterns stored in the repository. The Analyzer makes use of the monitoring policy to select the appropriate monitoring patterns for a particular process. Every policy is analysed according to the ConSpec specification, particular, if a policy has a Scope Session policy initialised when a service is invoked. The PDP helps in translating ConSpec policies into monitoring rules for decision making. 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. Note that if no guards resulted to "true" (and updates for ELSE are not specified), this means violation of the policy. If no updates are necessary for some conditions, a special command skip is envisaged.

Notification Module: It is developed as a part of the Aniketos platform and is used by the monitoring framework to report any violations. The Notification Module is implemented as a cloud service and is based on a publish-subscribe paradigm that notifies the entities subscribed about contract violation.

6 Conclusion

The presented monitoring framework is tightly integrated into the Aniketos platform (See Chapter 4) which supports the design-time and runtime aspects of secure and trustworthy service compositions. The 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 to understand and the simplicity of the language allows comparatively simple semantics. This enables the service designer to easily specify the monitoring requirements for their processes and monitor them using the framework. The monitoring framework is based on the way relevant information can be combined from multiple dynamic services in order to automate the monitoring of business processes and proactively report compliance violations. Alerts regarding policy violations are sent as notifications which other interested parties (generally the service composition providers) can subscribe to, allowing them to make verifications and take decisions and actions.

References

[1] Ghezzi, C., Guinea, S.: Run-time Monitoring in Service Oriented Architectures. In: Test and Analysis of Web Services. Springer, Heidelberg (2007)

[2] OMG, Business Process Model and Notation (BPMN) Version 2.0 (2011), `http://www.omg.org/spec/BPMN/2.0/`

[3] Rademakers, T.: Activiti in Action:Executable business processes in BPMN 2.0. Manning Publications (2012)

[4] Baresi, L., Guinea, S., Nano, O., Spanoudakis, G.: Comprehensive monitoring of BPEL processes. IEEE Internet Computing 14(3), 50–57 (2010)

[5] Haiteng, Z., Zhiqing, S., Hong, Z.: Runtime Monitoring Web Services Implemented in BPEL. In: International Conference on Uncertainty Reasoning and Knowledge Engineering (URKE), Bali, Indonesia, vol. 1, pp. 228–231 (2011)

[6] Wu, G., Wei, J., Huang, T.: Flexible Pattern Monitoring for WS-BPEL through Stateful Aspect Extension. In: Proc. of the IEEE Intl. Conf. on Web Services (ICWS 2008), Beijing, China, pp. 577–584 (2008)

[7] Baresi, L., Ghezzi, C., Guinea, S.: Smart Monitors for Composed Services. In: Proceedings of the 2nd International Conference on Service Oriented Computing (ICSOC 2004), New York, USA, pp. 193–202 (2004)

[8] Aniketos Consortium, Deliverable D9.2: Demonstration material and events from the complete project (2012)

[9] eXtensible Access Control Markup Language (XACML) Version 3.0, `http://docs.oasis-open.org/xacml/3.0/xacml-3.0-core-spec-os-en.pdf`

[10] Shanahan, M.: The Event Calculus Explained. In: Veloso, M.M., Wooldridge, M.J. (eds.) Artificial Intelligence Today. LNCS (LNAI), vol. 1600, pp. 409–430. Springer, Heidelberg (1999)

[11] Bonatti, P.A., De Coi, J.L., Olmedilla, D., Sauro, L.: PROTUNE: A Rule-based PROvisionalTrUst Negotia-tion Framework (2010)

[12] Aktug, I., Naliuka, K.: ConSpec: A Formal Language for Policy Specification. In: Proceedings of the First International Workshop on Run Time Enforcement for Mobile and Distributed Systems (2007)
[13] Erlingsson, U.: The inlined reference monitor approach to security policy enforcement. PhD thesis, Department of Computer Science, Cornell University (2004)
[14] Brucker, A.D.: Integrating Security Aspects into Business Process Models. IT - Information Technology 55(6), 239–246 (2013)
[15] S3MS project, `http://researchprojects.kth.se/index.php/kb_1/io_9718/io.html`
[16] Activiti engine, `http://www.activiti.org/`
[17] Aniketos Consortium, Deliverable D9.2: Demonstration material and events from the complete project (2012)