Optimizing Service Protection with Model Driven Security@run.time

Francis Wendpanga, Frédérique Biennier, Philippe Merle

Optimizing Service Protection with Model Driven Security@run.time

Wendpanga Francis Ouedraogo
Université de Lyon, CNRS INSA-Lyon,
LIRIS UMR 5205,
20 avenue Albert Einstein,
69621 Villeurbanne Cedex, France
wendpanga-francis.ouedraogo@liris.cnrsf.fr

Frédérique Biennier
Université de Lyon, CNRS INSA-Lyon,
LIRIS UMR 5205,
20 avenue Albert Einstein,
69621 Villeurbanne Cedex, France
frederique.biennier@liris.cnrsf.fr

Philippe Merle
Inria Lille - Nord Europe,
Parc Scientifique de la Haute Borne,
40 avenue Halley,
59650 Villeneuve d'Ascq, France
philippe.merle@inria.fr

***Abstract*—Enterprises are more and more involved in collaborative business. This leads to open and outsourcing all or part of their information system (IS) to create collaborative processes by composing business services picked in each partner IS and to take advantage of Cloud computing. Business services outsourcing and their dynamic collaboration context can bring lost of control on IS and new security risks can occur. This leads to inconsistent protection allowing competitors to access to unauthorized information. To address this issue, systematic security service invocations may be added, without paying attention to the business context leading to costly over protection. To address this issue, an adaptive security service model deployment is required to provide a business service consistent protection by taking into account the collaboration context (business service data criticity, partners involved in the collaboration, etc.), and the cloud deployment and execution environment. In this paper, we propose an adaptive security model based on MDS@run.time, the marriage of Model Driven Security (MDS) and Models@run.time approaches, allowing to select at runtime the appropriate security components to apply. The MDS approach is used to generate security policies, which are interpreted at runtime and load appropriate security mechanisms depending on the context (which takes advantage of the Models@run.time approach) ensuring business process end to end protection. A proof of concept prototype is built on top of the OW2 FraSCAti middleware, validating our proposition efficiency. Our experiments and simulations show that MDS@run.time improves the system efficiency when the over-protection risk rate increases.**

I. Introduction

Today Web 2.0 development calls for a new and distributed IT infrastructure to fit the cloud models to allow an efficient and distributed execution across the Web. As data and services hosted in a cloud are let out by their owner, new security requirements emerge in order to support a long-life due usage control. This trend is reinforced due to the lack of trust on such cloud organisation [1] and the rather poor adaptability level of the current security policies are often seen as braking forces to such XaaS developments.

Unfortunately, information system security economics have mostly been designed for well-perimetrised systems. Such security strategy is based on identifying threats and vulnerabilities and implementing counter-measures to reduce these risks. Different methods and tools (EBIOS, CERT/Octave, SNA, Safe/CISCO)[1] are available to implement this security strategy vision, i.e., facing identified vulnerabilities and reducing costs of security risks in a well-perimetrised and rather static environment. This static and perimetrised vision does not fit the multi-contextual execution environment involved by the service paradigm as a service can be invoked in different contexts. This may lead to inconsistent security deployment, under-protecting core information/process while running services in a new risky context or to an inefficient over-protection, locking the corporate information system access. Focusing on the service field, security has received a special attention over the last years [2]. Older works [3] focus on key technologies to support basic interoperable and standardised security services (mostly regarding transport security, message integrity and confidentiality, and even federation management to support cross authentication, etc.). The OASIS service reference model [4] takes advantage of the policy specification to trust management, authentication, authorisation and other security functions in the service model description but it does not allow to link these security features nor the security threats to the dynamic execution context of a service. This may lead to systematic and costly over protection when even useless security services are deployed or to risky under protection when useful security services are omitted.

Two other trends are interesting. Firstly, Model Driven Security (MDS) proposes to capture security policies as first-class models. MDS has already successfully applied to industrial systems [5], process-oriented systems [6], complex distributed systems [7], and multi-cloud context [8]. Nevertheless, MDS can not take into account runtime contexts because security policy models are only considered as design/development time entities. Secondly, Security as a Service (SecaaS) [9] considers security mechanisms as reusable runtime entities. Unfortunatly, the SecaaS approach is rarely implemented in a service-oriented way.

To overcome these limits, we propose a Model Driven Security@run.time approach that considers security policies as models interpreted at runtime [10] and that identifies the security requirements fitting the execution context. Plugged on the middleware used to implement the service system, our MDS@run.time component outsources the security management from the business service and provides an ad-hoc security

[1] https://www.enisa.europa.eu/activities/risk-management/current-risk/risk-management-inventory/rm-isms

mediation, i.e., select, compose and orchestrate security services (SecaaS) according to the execution context. A proof of concept has been developed on the OW2 FraSCAti middleware and used to evaluate our proposal. Our experiments and simulations show that MDS@run.time improves the system efficiency when the over-protection risk rate increases.

After presenting the context and state of the art (Section II), we introduce a motivating example (Section III) before presenting our MDS@run.time architecture (Section IV) and evaluating it (Section V) before presenting futher works (Section VI).

II. Context and State of the Art

Taking advantage of both the agility provided by the service selection/composition/orchestration mechanisms and of the interoperability provided by the systematic use of Web services related standards, collaborative business processes can be designed as a composition of different business services picked from the partner's own information system, challenging for a consistent protection fitting each partner own security requirements.

To this end, some security annotations on UML diagrams (such as the multi-purpose UMLSec [11] or the rather access control oriented SecureUML [12] domain specific languages) or BPMN diagrams [13] can be specified while designing a collaborative business process. As far as service-based business process implementation is concerned, these works have led to different frameworks such as OpenPMF [14] or SECTET [15] that take advantage of the Model Driven Security engineering [16] to generate security policies depending on the requirements associated to the business process model. Nevertheless, none of them support the full transformation process: While BPSec [17] is focused on the requirement engineering part, including CIM (Computation Independent Model) and PIM (Platform Independent Model) models, SECTET and OpenPMF provide PIM, PSM (Platform Specific Model) and code generation features. Moreover, the generation process is achieved according to a static environment vision (perimetrised process and well-known deployment platform), leading to define different policies depending on the business context. This complexifies the policy management and limits a consistent protection evolution as modifications are achieved locally.

To overcome this limit and provide a consistent protection fitting the different service invocation contexts, we believe that a service should not be duplicated and that its protection should be outsourced from the service body, i.e., defined as an associated protection policy enriched according to the different execution contexts. Such an approach involves to structure security policies depending on the protection services that must be fulfilled.

To this end, one can use the security service organisation proposed by OASIS in its service reference architecture [18] that allows to outsource security management from the business service implementation. As presented in Table II, the different protection services are split according to three implementation layers:

- The *network layer* refers to the communication infrastructure security risks mitigation (such as deny of service attacks).
- The *transport layer* refers to the communication channel used to exchange messages between services.
- The *application layer* is deployed on the top and includes access control, safe storage, data integrity, and non repudiation.

TABLE I. Protection services according to the implementation layers

Security Constraint	**Implementation Layer**	**Security Goal**	**Security Standard**
Confidentiality	Network	secure network infrastructure	IPsec for VPN
	Transport	secure communication channel, i.e. encrypt information	TLS/SSL
	Application	Secure storage, and message exchanged based encryption	WS-Security: XML-Encyption
Integrity	Application	signed data stored and exchanged	WS-Security: XML-Signature
Availability	Network	DoS protection via Firewall, IDS	
	Application	Management of QoS	BSLA (Business Service Level Agreement)

III. Motivating Example

The reusing ability provided by Web services, that can be selected, composed, and orchestrated in different contexts, challenges a strict deployment of security services to provide the convenient level of protection. This can lead to a costly over-protection. A motivating example could be provided by a service used to check mechanical specifications of a new product, as shown in Fig. 1. As data produced and acceded by this service have a high patrimonial value for the enterprise, then different protections can be deployed:

- Strong authentication to support specification traceability, i.e., knowing who has achieved/worked on the specifications.
- Restricted access control to allow only people from the enterprise or some authenticated partners to accede to this service.
- Cryptographic algorithm to protect exchanged data.

This service can be invoked in different workflows, as shown in Fig. 1:

- The corporate Computer Aid Design CAD (CAD) modelling system can invoke this service to check the intermediate specification consistency (75% of all the invocations),
- The Product Lifecycle Management (PLM) system can invoke this service to check the product information before integrating these data in its data base (10% of all the invocations),
- Other CAD modelling systems used by partners to exchange new requirements can also invoke this service to check the ordered specifications involved in a collaborative engineering project (15% of all the invocations).

Of course, depending on the invocation context of this service, some security constraints could be relaxed:

- For internal validation achieved via an invocation by the CAD modelling system, no traceability is needed.
- When the validation service is invoked via the PDM system, the operation must be logged.
- The corporate network is protected so data can be exchanged safely.
- External access to the corporate network is provided to employees and protected via a VPN.
- When a collaboration is set with a partner, this specification validation service can be invoked from the partner CAD modeller to check the requirements. In this case, the different actions must be logged. All the actions achieved via the service and restricting the access to only authorised persons.

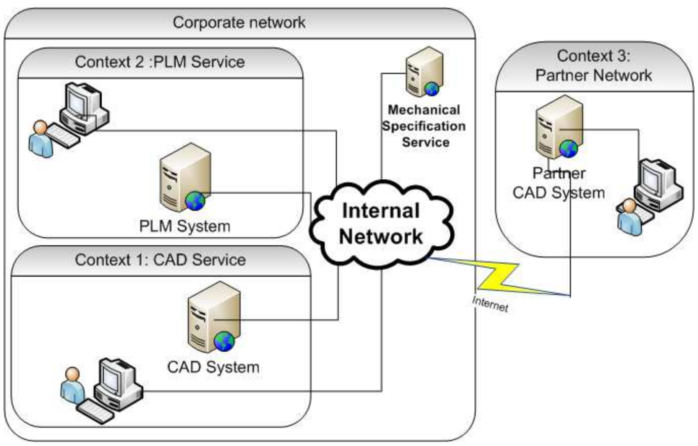

Fig. 1. Motivating example

These protection requirements lead to identify execution contexts for the specification validation service, as shown in Fig. 1:

- Context 1: Internal checking invoked by enterprise study board, thanks to their CAD Modelling System is a safe environment, no protection is required.
- Context 2: Certified requirement checking invoked by members of the enterprise from the PLM service to store checked specification. Authentication and non repudiation are required.
- Context 3: Collaboration specification checking invoked by a partner are from its CAD system to validate the order specification. Due to business constraints, authentication, authorization, and non repudiation are necessary whereas the opened execution platform requires data encryption.

```
1  <policies>
2    <policy id="1" resource="/mechanical/validateSpec/" type="Authentication" metric="
           0.25">
3      <policyRule>
4        <context type="uid/service" value="!{CADService}"/>
5        <pattern name="LoginPWD" metric="0.25">
6          <setting key="userRegistry" value="data/UserRegistry.xml"/>
7        </pattern>
8      </policyRule>
9    </policy>
10   <policy id="2" resource="/mechanical/validateSpec/" type="NonRepudiation">
11     <policyRule>
12       <context type="uid/service" value="!{CADService}"/>
13       <pattern name="log">
14         <setting key="file" value="data/log.log"/>
15       </pattern>
16     </policyRule>
17   </policy>
18   <policy id="3" resource="/mechanical/validateSpec/" type="Authorization">
19     <policyRule>
20       <context type="user/service" value="!{CADService,PDMService}"/>
21       <context type="Network/IP" value="![193.48.219.5,193.48.219.25]"/>
22       <context type="Network/DNS" value="!{mechanicalCompany.com}"/>
23       <pattern name="ACL">
24         <setting key="policyFile" value="acl/AccessControlList.xml"/>
25       </pattern>
26     </policyRule>
27   </policy>
28   <policy id="4" resource="/mechanical/validateSpec/" type="Confidentiality" metric=
           "0.75">
29     <policyRule>
30       <context type="user/service" value="!{CADService, PDMService}"/>
31       <context type="Network/IP" value="![193.48.219.5,193.48.219.25]" />
32       <context type="Network/DNS" value="!{mechanicalCompany.com}"/>
33       <pattern name="Encryption_AES_128" metric="0.75">
34         <setting key="EncryptionMethod" value="http://www.w3.org/2001/04/xmlenc#
                 aes128-cbc"/>
35         <setting key="part" value="body"/>
36       </pattern>
37     </policyRule>
38   </policy>
39 <policies>
```

Listing 1. Security policies associated to the *validateSpec* resource

Focusing on the mechanical validation specification service attached to the mechanical application, a global security policy can be set to define the different protection means to be deployed (see Listing 1) and their implementation context. This service includes an operation named *ValidationSpec*, which is considered as a resource (Line 2 in Listing 1). Its protection requires an authentication (Lines 2-9) if the caller service is not *CADService* (Line 4) using a simple login/password process (Line 5) refering to a checking file defined in Line 6. Besides authentication, it requires non repudiation feature (Lines 10-17) according to the service invoking it. Line 12 defines that except the interaction with *CADService* all services calling the *ValidateSpec* service have to be logged to ensure the traceability (Line 13). Besides non repudiation, according to the interaction with other services (Line 20), and to corporate internal network (Line 21) or the network domain (Line 22), access control rules have to be performed (Lines 18-27). ACL (Line 23) is the access control mechanism, which should be applied to this resource. Listing 2 describes the content of the authorization file allowing only *Service1* and *OtherService* to access the resource when the access is from an external network. Moreover the confidentiality (Lines 28-38) is required when the access to the resource is performed since an external network and the service invoking the resource is different from both corporate *CADService* and *PLMService*. In this context, the AES encryption (Lines 33-36) has to be applied during the exchanges with the resource.

```
<acl xsi:noNamespaceSchemaLocation="AccessListSchema.xsd">
  <resource name="/mechanical/validateSpec/">
    <grant user="OtherService" ></grant>
    <grant user="Service1"></grant>
  </resource>
</acl>
```

Listing 2. *AccessControlList.xml* authorization file

As this business service can be invoked dynamically by ad-hoc collaborative workflows, protection services must be deployed according to the execution context paying attention on both organisational (i.e., which partner, trusted or not, invokes the service) and technical (which kind of cloud hosts the collaborative workflow, which kind of transport service is provided, etc.) environments. The protection requirements are defined globally in the security policies attached to the different business services (see Listing 3, Line 3). These security policies are seen as security models at runtime that will be used at runtime by the security mediator to select, compose and orchestrate the security services depending on the execution context.

```
<wsdl:binding name="mechanicalServiceSoapBinding" type="
        tns:mechanicalServicePortType">
  <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
  <wsdl:operation name="validateSpec" mds:policyRef="data/policies.xml" xmlns:mds="
        http://mds.org">
    <soap:operation soapAction="" style="document"/>
    <wsdl:input name="validateSpec">
      <soap:body use="literal"/>
    </wsdl:input>
    <wsdl:output name="validateSpecResponse">
      <soap:body use="literal"/>
    </wsdl:output>
  </wsdl:operation>
</wsdl:binding>
```

Listing 3. Link the security policy with the *ValidateSpec* operation of the *Mechanical* system service

In our example, the *ValidationSpec* service and the related information must be protected in the different contexts. The confidentiality requirement impacts both application layer, which is in charge of the access control, traceability, and transport layer (see Listing 1). The systematic composition of the authorization, traceability and confidentiality services may be costly. Table II shows a comparison of service execution time according to the context (Network Internal/External used to interact with the service, and the services attended to the collaboration). Here *CADService* invokes *ValidateSpec* without any security implementation. *PLMService* can invoke the *ValidateSpec* service with the tracability requirement. For *other services*, in addition to the tracability requirement, both authorization and encryption are required to invoke *ValidateSpec*. Testing conditions and environment are detailed in Section V.

TABLE II. SERVICE EXECUTION TIME INCLUDING CONTEXT IDENTIFICATION

Context	Security services involved*ValidateSpec*	Execution time (ms)
Context 1	No security mechanism is required	64
Context 2	Authentication and Non Repudiation	80
Context 3	Context 2 security services + Authorization and Confidentiality	86

To avoid this costly over protection or risky under protection depending on the runtime environment vulnerability, we propose to turn these security policies as Models@run.time so that they can be analysed to select, compose and orchestrate the most convenient security services depending on the exact runtime environment. This requires a new architecture to *outsource* the security management as a new high-level service that can be plugged on the hosting middleware.

IV. MODEL DRIVEN SECURITY@RUN.TIME

Context-aware security management has lead to different works for more than the decade from the early adaptation of access control rules [19] to SLA and policy orchestration integration in cloud [20]. Nevertheless, these works do not fit the service dynamic reusing abilities and different policies are attached to each execution context. To overcome this limit, we propose to extend the Model Driven Security approach to a MDS@run.time strategy in order to capture the execution context while deploying the security services. This execution context integrates the business process related context defined by OASIS and information related to the execution platform. By this way, a single security policy gathers all the protection requirements and deployment can be outsourced from the business service and managed in a context-aware security strategy. We first present our context model before detailing our architecture.

A. Execution context specification

The execution context is defined as a set of functional, organisational and technological specifications, which determine the choice of security measures to perform. The functional specification allows knowing the types of information (strategic, personal, financial, etc.) used or handled by each service and the information sensitive level (top secret, secret, confidential, restricted, unclassified). This specification aims to know which protection and level of protection are required in terms of confidentiality and/or integrity. Organizational specification focused on access control and other security criteria such as availability and non-repudiation. It allows to answer to the questions: Who can access and interact with process resources? By using which means, such as networks, devices, etc.? From which locality and at which time? Moreover this organization specification allows defining both obligation and restriction constraints in terms of access control. Based on these risk-analysis specification, a consistent security policy can be defined for each service gathering the different protection services. As show in TABLE II, this can be rather costly. As a consequence, we integrate contextual information to motivate the use of the different security assertions depending of the context. To this end, our formal context model is defined by different parameters:

- *Who*: Refer to the business service invoking the protected service.
- *For who*: Specify the user for whom the service is invoked.
- *From where*: Specify the network specification.
- *When*: Define time depending of the protection management.

Thanks to this extended context specification, one can select the security assertion to be composed and deployed at runtime.

B. MDS@run.time architecture

To deploy our adaptive security model [21], we take advantages of the SOA distributed implementation. Here an SOA middleware plays an intermediary role between the client and the service provider. It is a software component, which is located between the operating system and business applications, and offers a high level abstraction for building distributed applications. It allows business service integration and management, and provides access to various external services. It is an integration solution, which implements a fully distributed architecture deployed on multiple nodes, providing services such as data processing or Content Based Routing (CBR), and a higher level of interoperability by systematically using standards such as XML, WS-* specifications [3].

Our outsourced context-aware security architecture [22] is plugged on the middleware capturing the service invocation (see Fig. 2). A middleware specific interceptor intercepts the service invocation (Step 1) and routes this request to the MDS@run.time component (Step 2). Based on the request, the

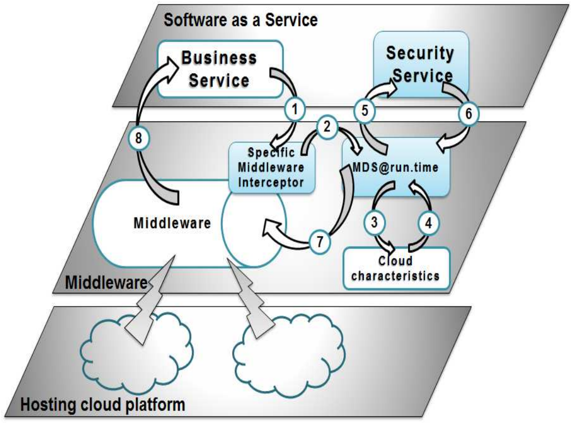

Fig. 2. MDS@run.time architecture

MDS@run.time component retrieves all the policies associated to the invoked business service. Thanks to the environment characteristics (cloud platform, devices and network used, etc.) (Steps 3 and 4), only the security policies matching the current context are selected and composed to implement the required protection. Then the MDS@run.time component orchestrates the security service invocations (Steps 5 and 6) by using the security as a service component, which implements the security mechanisms defined in each policy (Steps 5 and 6). If succeeded, the invoker is granted and security mechanisms are applied, the MDS@run.time component routes back the business service/middleware (Steps 7 and 8). But in unsuccessful case, the business service is not invoked and an error message is returned to the invoker.

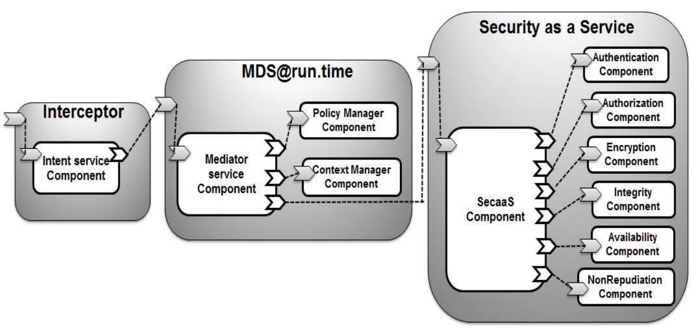

Fig. 3. MDS@run.time components

The MDS@run.time architecture is composed of three components as illustrated in Fig. 3: `Interceptor`, `MDS@run.time`, and `Security as a Service`. This design allows us to clearly separate concerns: specific middleware interception, MDS@run.time mediation, and security services. Moreover, this design provides portability of both `MDS@run.time` and `Security as a Service` components across different middleware platforms.

The specific middleware interceptor is defined to capture and route the interactions between the business service and the middleware to our MDS@run.time component. But as how request interception is done is specific to each middleware, this component must be specifically implemented for each middleware, see Section V-A for details on how this component is implemented for the OW2 FraSCAti middleware.

The MDS@run.time component is the core element of our context-aware architecture [22] and is in charge of identifying the assertion to implement, composing and orchestrating the security services accordingly. It includes three components as illustrated in Fig. 3:

- The security `mediator` is the entry point of this component. It analyses the interaction received from the interceptor that defines the invoked service and the execution environment context information identifying some context parameters as:
  - Who, i.e, the service invoking the business service.
  - When, i.e., capture the invocation time.
  - From where, i.e, capture the geographic location of the calling service.

  Then, it invokes the policy manager to extract the policy associated to the invoked service and the context manager, which will compose and deploy the security services depending on the context.
- The `PolicyManager` component manages the security policies. It receives from the `Mediator` the resource or service reference requested and the link to the policy file. It returns to the `Mediator` the list of security policies to apply.
- The `ContextManager` component analyses security policies associated to services and identifies the different policies to be applied according to the user context, the execution environment and security policies associated to the client and service provider. It also provides to the `Mediator` component information such as policies and policy rules related to the execution context. These policy rules are used by the `Mediator` component to call the technical security services.

To support a fully outsourced security strategy, this architecture is enriched with a Security as a Service (`SecaaS`) component, which gathers implementations of the different security services based on standards such as:

- The `Authentication` component is used to prove the user identity (of human or other service). This component receives from the `SecaaS` component the policy rule to apply, extracts information about the security pattern and invokes the security mechanism to be applied. It can be a weak authentication mechanism such as login/password or strong authentication such as One Time Password (OTP) or two factors authentication. This `Authentication` component includes subcomponents such as the `SSORegistry` (Single Sign On Registry) component used to store information about authentication of sessions and to allow to retrieve user information without restarting authentication.
- The `Authorization` component manages access to resources and services, and allows grant or deny the user access to them. As the `Authentication` component, it receives the security policy rule and invokes the authorization mechanism to be applied. This mechanism can be based on an authorization by role

(RBAC) implemented by the XACML authorization protocol or a simple Access Control List (ACL).

- The `Encryption` component provides data and messages encryption/decryption mechanisms. It also provides secure protocols using secure communication (SSL).
- The `Integrity` component ensures the integrity of exchanged data and messages by using message signatures or hash functions.
- The `NonRepudiation` component is responsible for recording user actions (authentication, access to data or service, data modification/destruction, etc.). This information can then be used for auditing and monitoring.
- The `Availability` component is responsible for the services' availability providing access to the service or a clone (redundant service) thereof if the original target service is unavailable. This component also provides backup mechanism to restore system data and services after disaster.

The `Encryption`, `Integrity` and `NonRepudiation` components can use security protocols such as WS-Security XML Encryption and XML Signature, which provide encryption and signing exchanged message mechanisms.

V. EVALUATION

Our MDS@run.time architecture is designed to support an outsourced security deployment. As presented in Fig. 2, it is designed in a non intrusive way for both services and middleware. To evaluate our proposal, we present in the following sub-sections a Proof of Concept prototype plugged on the OW2 FraSCAti middleware before evaluating its performance level on the motivating example presented in Section III, and comparing the results with other approaches to manage security in collaborative processes.

A. Organisation of the Proof of Concept prototype

FraSCAti[2] [23] is an open source middleware framework to build, develop, deploy, execute, and manage adaptable service-oriented business applications. FraSCAti is based on the OASIS Service Component Architecture (SCA) standard[3]. FraSCAti applications can be deployed in different clouds such as Amazon EC2, Appfog, CloudBees, DELL KACE, dotCloud, Eucalyptus, Jelastic, Heroku, OpenShift, Windows Azure, as discussed in [24], [25], [26]. The adaptability at design time is based on the fact that the FraSCAti platform was designed as a plugin-based architecture to adapt it to different execution environments and to select on demand the required application functionalities composing a FraSCAti instance [27]. The adaptability at execution time is based on the FraSCAti reflective features, which encompass introspection and reconfiguration of applications at runtime [28]. For dealing with web services and REST, FraSCAti embeds Apache CXF[4], a well-known open source services framework. To ensure the business process security deployed on cloud infrastructures, we propose a MDS@run.time framework based on SCA components, which can be plugged to the FraSCAti platform. Our prototype takes advantage of Aspect Oriented Programming (AOP [29]) features and of the SCA model, both provided by FraSCAti, to deploy the three `Interceptor`, `MDS@run.time` and `Security as a Service` components shown in Fig. 3.

SCA provides the notion of intent, which is an abstraction for designating a non-functional property such as security, transaction, logging, etc. With FraSCAti, SCA intents are implemented as SCA components, then both business and non-functional concerns are designed then implemented in the same framework, aka SCA.

The `Intent` component is responsible for detecting and intercepting business services invoked by clients. This component uses AOP techniques provided by FraSCAti to perform actions before, during and after each business service invocation. These techniques use the Apache CXF interception mechanism. The `Intent` component creates a `Request` object, which plays the intermediary role between the FraSCAti middleware and security services. This object provides a bidirectional interface that allows the `Intent` component to formalize the interaction messages received from Apache CXF and also to specify orders towards Apache CXF. The `Request` object ensures a total independence between our MDS@run.time component and the underlying service-oriented middleware, allowing on one hand the security services to be able to deploy and run on any other middleware and on another hand to deploy on a specific platform just the required security services.

In our MDS@run.time prototype, we reused and enhanced the Enterprise Java XACML framework[5]. The data encryption function is AES 128-bit. Authentification by login/pwd and non-repudiation by logs are implemented in an ad hoc way. For authentification, unique tokens are generated to avoid to manually re-authentifying when accessing other services (SSO). As the thread model of Apache CXF is to affect a thread to each incoming request, then potentially concurrent threads can execute our security services. The MDS@run.time security services must then be protected against concurrent accesses via critical sections like Java monitors.

B. Performance evaluation

Our performance evaluation is based on the use case presented in Section III, focusing on the mechanical system *ValidationSpec* operation. This operation is implemented thanks to a service associated to a security policy including non repudiation (see Listing 1, Lines 2-9), access control (Lines 10-17) using ACL (Lines 23-25) and confidentiality (Lines 28-38). As far as the collaborative service is concerned, the business service is encapsulated in a *MechanicalService*, which is associated to the convenient security policy and refers to the `MDS@run.time` composite (Listing 4, Line 4). By this way, the business service can be intercepted and MDS@run.time is invoked before invoking the business service itself.

```
1 <composite name="Mechanical" >
2   <service name="Mechanical" promote="MechanicalComponent/Converter">
3     <interface.java interface="mechanical.api.mechanicalService"/>
```

[2]http://frascati.ow2.org

[3]http://www.oasis-opencsa.org/sca

[4]http://cxf.apache.org

[5]https://code.google.com/p/enterprise-java-xacml/

```
<binding.ws requires="MDSatRuntime" uri="/mechanical-ws-mds" wsdlElement="
        http://api.mechanical/#wsdl.port(mechanicalService/
        mechanicalSpecServicePort)" wsdli:wsdlLocation="wsdl/mechanical.wsdl"/>
  <frascati:binding.rest requires="MDSatRuntime" uri="/mechanical-rest-mds"/>
 </service>   .
</composite>
```

Listing 4. Link the *Mechanical* component with **MDS@run.time**

To evaluate the impact of our *MDS@run.time with FraSCAti* prototype on the service execution time, we set a benchmarking environment using FraSCAti version 1.6 with Oracle Java Virtual Machine 1.7.0_51 on Microsoft Windows 7 Professional (32 bit) using a 2,54GHz processor Intel(R) Core(TM)2 Duo CPU with 4Go of memory.

Firstly, we measured the execution time of each component: The business service without invoking our security architecture (Measure 1 in TABLE III), the FraSCAti service interception (Measure 2), the `mediator` component (Measure 3), the authentication service (Measure 4), the non repudiation service (Measure 5), the authorization service (Measure 6), and the confidentiality service (Measure 7). We manage a benchmark loop to compute an average execution time on 1000 client requests, so that extra factor impacts, such as bootstrapping effects, Just-In-Time compilation, etc., can be smoothed.

TABLE III. EXECUTION TIME OF **MDS@RUN.TIME** COMPONENTS

No	Component	Average execution time (ms)	Systematic protection	Context 1	Context 2	Context 3
1	FraSCAti + Apache CXF + Business service	58	58	58	58	58
2	FraSCAti Interceptor	2		2	2	2
3	MDS@run.time	4		4	4	4
4	Authentication	4	4		4	4
5	NonRepudiation	12	12		12	12
6	Authorization	2	2			2
7	Confidentiality	4	4			4
	Total		80	64	80	86

The execution meantime for a systematic protection, i.e., all security services are invoked but without our MDS@run.time components, is 80 milliseconds (Column 4 in Table III). The execution meantime for Context 1, involving our MDS@run.time components but no security services, is 64 milliseconds, i.e., the business service execution time plus the MDS@run.time overhead. The execution meantime for Context 2, involving MDS@run.time plus authentification and non-repudiation services, is 80 milliseconds. The execution time of Context 3, involving all security services, is 86 milliseconds.

So, the first result is that the interception and mediation process (Measure 2 plus 3 in Table III) represents only 6 milliseconds, i.e., around 7% of the total execution time of Context 3. This overhead could certainly reduced within an industrial implementation of MDS@run.time by for instance merging/moving the implementation code of both `MDS@run.time` and `Security as a Service` components into the interceptor. But then we will lose the portability of these components on different middleware platforms. Then this is an implementation trade-off between performance and portability. However this demonstrates that our MDS@run.time approach, i.e., interpretation of security policies at runtime, introduces a small overhead compared to a systematic deployment of the different security services.

Table IV reports the unitary execution time and the total execution time split according to the three contexts paying attention of the occurrence rate of each context (see Section II). It shows that our security adjustment allows to reduce the total execution time for about 14% compared to the systematic protection (over-protection) scenario.

TABLE IV. EXECUTION TIMES EVALUATION

	With our approach			Systematic protection
	Context 1	Context 2	Context 3	
Execution time for one invocation (ms)	64	80	86	80
Rate of 1000 invocations	75%	10%	15%	100%
Total execution of each context	48 000	8 000	12 900	80 000
Total of execution time for 1000 invocation (ms)	**68 900**			**80 000**

We extend this benchmark to integrate business services that request longer or smaller computation times (see Table V) to compare the cost of context analysis and of the dynamic security composition and orchestration at runtime. Our simulation uses several business services whose execution times vary from 10 ms to 100 ms. We set 2 reference execution contexts, one required no security deployment (Context 1) whereas the other requires the maximum protection (i.e systematic protection required in Context 3 associated to our motivating example). Measures are achieved using different rates for Context 1 (from 10% to 100%). The results show that the MDS@run.time maximum cost varies from 18,75% of the smaller service execution time to 4,9% for the bigger one when a systematic protection is required. On the opposite, MDS@run.time exhibits a benefit of 13% to 50% of the execution time when no protection is required. These results show that the overhead involved by our MDS@run.time architecture can be rather neglected compared to the large overhead introduced by the systematic invocation of (often) useless security services provided that the no protection rate is greater than 30% as shown in Fig. 4.

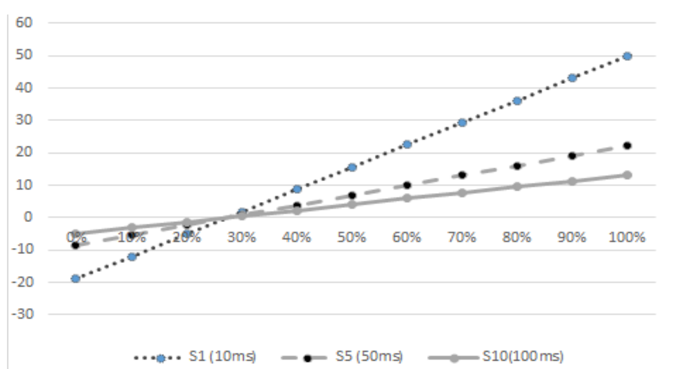

Fig. 4. Variation of MDS@run.time execution cost according to three business services

C. Comparison with other related works

As stated in Section II, different works on Model Driven Security [16] have been conducted to integrate security annota-

TABLE V. EXECUTION TIMES FOR A PANEL OF 1000 INVOCATIONS OF BUSINESS SERVICES WHERE THE NO PROTECTION RATE EVOLVES FROM 0% TO 100%

	Systematic protection	0%	10%	20%	30%	40%	50%	60%	70%	80%	90%	100%
S1 (10ms)	32000	38000	35800	33600	31400	29200	27000	24800	22600	20400	18200	16000
S2 (20ms)	42000	48000	45800	43600	41400	39200	37000	34800	32600	30400	28200	26000
S3 (30ms)	52000	58000	55800	53600	51400	49200	47000	44800	42600	40400	38200	36000
S4 (40ms)	62000	68000	65800	63600	61400	59200	57000	54800	52600	50400	48200	46000
S5 (50ms)	72000	78000	75800	73600	71400	69200	67000	64800	62600	60400	58200	56000
S6 (60ms)	82000	88000	85800	83600	81400	79200	77000	74800	72600	70400	68200	66000
S7 (70ms)	92000	98000	95800	93600	91400	89200	87000	84800	82600	80400	78200	76000
S8 (80ms)	102000	108000	105800	103600	101400	99200	97000	94800	92600	90400	88200	86000
S9 (90ms)	112000	118000	115800	113600	111400	109200	107000	104800	102600	100400	98200	96000
S10(100ms)	122000	128000	125800	123600	121400	119200	117000	114800	112600	110400	108200	106000

TABLE VI. VARIATION OF MDS@RUN.TIME EXECUTION COST COMPARED TO BUSINESS SERVICES EXECUTION TIMES DEPENDING ON "THE NO PROTECTION" CONTEXT RATE

Business services	0%	10%	20%	30%	40%	50%	60%	70%	80%	90%	100%
S1 (10ms)	-18,75%	-11,88%	-5,00%	1,88%	8,75%	15,63%	22,50%	29,38%	36,25%	43,13%	50,00%
S2 (20ms)	-14,29%	-9,05%	-3,81%	1,43%	6,67%	11,90%	17,14%	22,38%	27,62%	32,86%	38,10%
S3 (30ms)	-11,54%	-7,31%	-3,08%	1,15%	5,38%	9,62%	13,85%	18,08%	22,31%	26,54%	30,77%
S4 (40ms)	-9,68%	-6,13%	-2,58%	0,97%	4,52%	8,06%	11,61%	15,16%	18,71%	22,26%	25,81%
S5 (50ms)	-8,33%	-5,28%	-2,22%	0,83%	3,89%	6,94%	10,00%	13,06%	16,11%	19,17%	22,22%
S6 (60ms)	-7,32%	-4,63%	-1,95%	0,73%	3,41%	6,10%	8,78%	11,46%	14,15%	16,83%	19,51%
S7 (70ms)	-6,52%	-4,13%	-1,74%	0,65%	3,04%	5,43%	7,83%	10,22%	12,61%	15,00%	17,39%
S8 (80ms)	-5,88%	-3,73%	-1,57%	0,59%	2,75%	4,90%	7,06%	9,22%	11,37%	13,53%	15,69%
S9 (90ms)	-5,36%	-3,39%	-1,43%	0,54%	2,50%	4,46%	6,43%	8,39%	10,36%	12,32%	14,29%
S10(100ms)	-4,92%	-3,11%	-1,31%	0,49%	2,30%	4,10%	5,90%	7,70%	9,51%	11,31%	13,11%

tions in the process model specification. Nevertheless, none of them support the full transformation process nor the integration of multiple and dynamic contexts. This can lead to inconsistent protection as each security policy is generated separately. Table VII gives a detailed comparison between three state-of-the-art MDS frameworks and our MDS@run.time framework.

TABLE VII. COMPARISON BETWEEN MDS FRAMEWORKS FOR SERVICE-ORIENTED SYSTEMS

MDS Framework	OpenPMF [14]	SECTET [15]	BPSec [17]	MDS@ run.time
Abstraction level				
CIM	No	No	Yes	Yes
PIM	Yes	Yes	Yes	Yes
PSM-code	Yes	Yes	No	Yes
Support of				
Authentification	Yes	Yes	No	Yes
Authorisation	Yes	Yes	Yes	Yes
Integrity	No	Yes	No	Yes
Encryption	No	Yes	No	Yes
Non repudiation	Yes	Yes	Yes	Yes
Availability	No	No	No	Yes
Privacy	No	No	Yes	Yes
Support of				
SAML	No	Yes	No	Yes
XACML	Yes	Yes	No	Yes
WS-Policy	No	Yes	No	Yes
WS-Security	No	No	No	Yes
Consideration of				
infrastructure	No	No	No	Yes
runtime context	No	No	No	Yes
SecaaS	No	Yes	No	Yes
MDS@run.time	No	No	No	Yes

Our MDS@run.time vision overcomes these limits as:

- All requirements are gathered in a single protection policy attached to the business service, which policy is generated thanks to a fully automatised generation process.
- Security deployment is outsourced from the business process orchestration process as the security policy is analysed dynamically according to the technological and organisational execution context.
- The security architecture is designed in a non-intrusive way and can fit multi-cloud deployment as it is plugged on the service middleware.

Moreover, our context definition enriches the context model proposed in the OASIS reference architecture with technical context information picked from the different cloud security models. So services can be secured on the fly. Thanks to the execution platform information, collected by the `mediator` component, security services are selected, composed and orchestrated in a transparent and consistent way, avoiding the costly over-protection and the risky under protection.

VI. CONCLUSION

Securing collaborative business processes deployed on cloud systems require paying attention on both organisational and platform-related vulnerabilities. Taking advantage of the intrinsic flexibility provided by the association of security policies to services, we propose to use them as Models@run.time to select, compose and orchestrate security services depending on the required protection and on the execution context. To this end, a MDS@run.time component is plugged on the middleware, intercepting service invocation and capturing context information. The experiment reported in this paper shows how our MDS@run.time architecture can be plugged on the OW2 FraSCAti middleware and reduce the execution time compared to a systematic over-protection approach. Further works will focus on the integration of more detailed platform models and on vulnerability monitoring loops so that our coarse-grained vision of the execution context will be refined to increase the protection efficiency.

REFERENCES

[1] H. Jay and M. Nicolett, "Assessing the security risks of cloud computing," *Gartner Report*, 2008.

[2] L. Martino and E. Bertino, "Security for Web Services: Standards and Research Issues," *Int. J. Web Service Res.*, vol. 6, no. 4, pp. 48–74, 2009. [Online]. Available: http://dx.doi.org/10.4018/jwsr.2009071303

[3] IBM and Microsoft Corp, "Security in a Web Services World : a proposed architecture and roadmap," 2002. [Online]. Available: ftp://www6.software.ibm.com/software/developer/library/ws-secmap.pdf

[4] Organization for the Advancement of Structured Information Standards (OASIS), "Reference Architecture Foundation for Service Oriented Architecture@ONLINE," pp. 96–102, october 2009. [Online]. Available: http://docs.oasis-open.org/soa-rm/v1.0/soa-rm.html

[5] M. Clavel, V. Silva, C. Braga, and M. Egea, "Model-Driven Security in Practice: An Industrial Experience," in *4th European Conference on Model Driven Architecture: Foundations and Applications (CMDA-FA 08)*, 2008, pp. 326–337.

[6] D. Basin, J. Doser, and T. Lodderstedt, "Model Driven Security for Process Oriented Systems," in *8th ACM Symposium on Access Control Models and Technologies (SACMAT 03)*. ACM, 2003, pp. 100–109.

[7] U. Lang and R. Schreiner, "Model Driven Security Management: Making Security Management Manageable in Complex Distributed Systems," in *Workshop on Modeling Security (MODSEC08) - International Conference on Model Driven Engineering Languages and Systems (MODELS)*, 2009.

[8] W. F. Ouedraogo, F. Biennier, and P. Ghodous, "Model Driven Security in a Multi-Cloud Context," *International Journal of Electronic Business Management*, vol. 11, no. 3, pp. 178–190, 2013.

[9] Cloud Security Alliance, "Security Guidance for Critical Areas of Focus in Cloud Computing- CSA Guidance v3," Jun. 2011. [Online]. Available: http://www.cloudsecurityalliance.org/guidance/csaguide.v3.0.pdf

[10] G. Blair, N. Bencomo, and R. B. France, "Models@run.time," *Computer*, vol. 42, no. 10, pp. 22–27, 2009.

[11] J. Jürjens, "UMLsec: Extending UML for Secure Systems Development," in *Proceedings of the 5th International Conference on the Unified Modeling Language*, ser. UML '02. London, UK, UK: Springer-Verlag, 2002, pp. 412–425. [Online]. Available: http://dl.acm.org/citation.cfm?id=647246.719625

[12] T. Lodderstedt, D. A. Basin, and J. Doser, "SecureUML: A UML-Based Modeling Language for Model-Driven Security," in *Proceedings of the 5th International Conference on the Unified Modeling Language*, ser. UML '02. London, UK, UK: Springer-Verlag, 2002, pp. 426–441. [Online]. Available: http://dl.acm.org/citation.cfm?id=647246.719477

[13] C. Wolter, M. Menzel, A. Schaad, P. Miseldine, and C. Meinel, "Model-driven business process security requirement specification," *Journal of Systems Architecture (JSA)*, pp. 211–223, 2009.

[14] U. Lang, "OpenPMF SCaaS: Authorization as a Service for Cloud & SOA Applications," in *2010 IEEE Second International Conference on Cloud Computing Technology and Science (CloudCom)*, Nov 2010, pp. 634–643.

[15] M. Alam, M. Hafner, and R. Breu, "Constraint based role based access control in the SECTET-framework A model-driven approach," *Journal of Computer Security*, pp. 223–260, 2008.

[16] L. Lcio, Q. Zhang, P. H. Nguyen, M. Amrani, J. Klein, H. Vangheluwe, and Y. L. Traon, "Chapter 3 - Advances in Model-Driven Security," in *Advances in Computers*, A. Memon, Ed. Elsevier, 2014, vol. 93, pp. 103 – 152. [Online]. Available: http://www.sciencedirect.com/science/article/pii/B9780128001622000038

[17] A. Rodríguez, E. Fernández-Medina, and M. Piattini, "A BPMN Extension for the Modeling of Security Requirements in Business Processes," *IEICE - Trans. Inf. Syst.*, vol. E90-D, no. 4, pp. 745–752, Mar. 2007. [Online]. Available: http://dx.doi.org/10.1093/ietisy/e90-d.4.745

[18] Organization for the Advancement of Structured Information Standards (OASIS), "Reference Model for Service Oriented Architecture 1.0: OASIS Standard@ONLINE," october 2006. [Online]. Available: http://docs.oasis-open.org/soa-rm/v1.0/soa-rm.html

[19] M. J. Covington, W. Long, S. Srinivasan, A. K. Dev, M. Ahamad, and G. D. Abowd, "Securing Context-aware Applications Using Environment Roles," in *Proceedings of the Sixth ACM Symposium on Access Control Models and Technologies*, ser. SACMAT '01. New York, NY, USA: ACM, 2001, pp. 10–20. [Online]. Available: http://doi.acm.org/10.1145/373256.373258

[20] M. C. Mont, K. McCorry, N. Papanikolaou, and S. Pearson, "Security and Privacy Governance in Cloud Computing via SLAs and a Policy Orchestration Service," in *CLOSER 2012 - Proceedings of the 2nd International Conference on Cloud Computing and Services Science, Porto, Portugal, 18 - 21 April, 2012*, 2012, pp. 670–674.

[21] W. F. Ouedraogo, F. Biennier, and P. Ghodous, "Adaptive Security Policy Model to Deploy Business Process in Cloud Infrastructure," in *2nd International Conference on Cloud Computing and Services Science (CLOSER 2012)*, 2012, pp. 287–290.

[22] W. F. Ouedraogo, F. Biennier, and P. Merle, "Contextualised security operation deployment through MDSrun.time architecture," in *ISC 2014 - Intelligent Service Clouds Workshop at the 12th International Conference on Services Oriented Computing 2014*, Paris, France, Nov. 2014, to appear. [Online]. Available: https://hal.archives-ouvertes.fr/hal-01088034

[23] L. Seinturier, P. Merle, D. Fournier, N. Dolet, V. Schiavoni, and J.-B. Stefani, "Reconfigurable SCA Applications with the FraSCAti Platform," in *IEEE International Conference on Services Computing (SCC'09)*. IEEE, 2009, pp. 268–275.

[24] P. Merle, R. Rouvoy, and L. Seinturier, "A Reflective Platform for Highly Adaptive Multi-Cloud Systems," in *International Workshop on Adaptive and Reflective Middleware (ARM'11) - 12th ACM/IFIP/USENIX International Middleware Conference*. ACM, 2011, pp. 14–21.

[25] F. Paraiso, N. Haderer, P. Merle, R. Rouvoy, and L. Seinturier, "A Federated Multi-Cloud PaaS Infrastructure," in *5th International Conference on Cloud Computing (CLOUD'12)*. IEEE, 2012, pp. 392–399.

[26] F. Paraiso, P. Merle, and L. Seinturier, "soCloud: A service-oriented component-based PaaS for managing portability, provisioning, elasticity and high availability across multiple clouds," *Computing*, vol. Special Issue on Cloud Computing, 2014, to appear.

[27] M. Acher, A. Cleve, P. Collet, P. Merle, L. Duchien, and P. Lahire, "Reverse Engineering Architectural Feature Models," in *Proceeding of 5th European Conference of Software Architecture, ECSA 2011*, ser. Lecture Notes in Computer Science (LNCS). Essen, Germany: Springer, Sep. 2011, vol. 6903, pp. 220–235. [Online]. Available: https://hal.inria.fr/inria-00614984

[28] L. Seinturier, P. Merle, R. Rouvoy, D. Romero, V. Schiavoni, and J.-B. Stefani, "A component-based middleware platform for reconfigurable service-oriented architectures," *Software: Practice and Experience*, vol. 42, no. 5, pp. 559–583, 2012.

[29] G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C. Lopes, J.-M. Loingtier, and J. Irwin, *Aspect-oriented programming*. Springer, 1997.