Contextualised security operation deployment through MDS@run.time architecture

Wendpanga Francis Ouedraogo, Frédérique Biennier, Philippe Merle

Contextualised security operation deployment through MDS@run.time architecture

Wendpanga Francis Ouedraogo[1], Frédérique Biennier[1], and Philippe Merle[2]

[1] Université de Lyon, CNRS INSA-Lyon, LIRIS UMR 5205, 20 avenue Albert Einstein, 69621 Villeurbanne Cedex, France
{wendpanga-francis.ouedraogo,frederique.biennier}@liris.cnrsf.fr
[2] Inria Lille - Nord Europe, Parc Scientifique de la Haute Borne, 40 avenue Halley, 59650 Villeneuve d'Ascq, France
philippe.merle@inria.fr

Abstract. The fast development of Cloud-based services and applications have a significant impact on Service Oriented Computing as it provides an efficient support to share data and processes. The deperimeterised vision involved by these Intelligent Service Clouds lead to new security challenges: providing a consistent protection depending on the business environment conditions and on the deployment platform specific threats and vulnerabilities. To fit this context aware protection deployment challenge, we propose a MDS@run.time architecture, coupling Model Driven Security (MDS) and Models@run.time approaches. By this way, security policies (that can be generated via a MDS process) are interpreted at runtime by a security mediator depending on the context. This proposition is illustrated thanks to a proof of concept prototype plugged on top of the FraSCAti middleware.

1 Introduction

The fast development of Cloud-based services and applications provides an efficient support to share data and processes, leading to deperimeterised Information Systems. The flexibility and agility provided by these so-called Intelligent Service Clouds enables new styles of inter-enterprises Collaborative Business, taking advantage of service reusability to create new collaborative workflows and of the Cloud plasticity allowing to use different access devices. This deperimeterised and evolving vision of Information Systems leads to enforce the call for protection mechanisms to mitigate potential vulnerabilities or threats related to any potential business (i.e. the specification of the organizations and workflow in which the service may be involved) or deployment context (namely access device, interconnection network or deployment platform configuration information). To avoid a systematic and costly over-protection deployment, we propose a context-aware security architecture to select at runtime the security policy rules that fit the current business and technical execution context. To this end, we couple the MDS [6] and Models@run.time approaches to set a MDS@run.time architecture. In brief, security policies (defining the different protection services depending on environmental conditions) are seen as an abstract Models@run.time specification used to select, compose and orchestrate security services depending on the current execution context. This architecture plugged on hosting middleware allows outsourcing security concerns from the business services. The service invocation is captured and processed to compose and orchestrate the convenient security services depending on the execution context. We call this process Security Mediation as it is built in a similar perspective of the classical outsourced mediation components. A proof of concept prototype plugged on FraSCAti is used to evaluate the "execution cost" of the dynamic security deployment. We first present the context and a motivating example before defining the way MDS is extended in a MDS@run.time vision (Section 2). The related architecture and its implementation plugged on the FraSCAti middleware is detailed in Section 3 and its performance are evaluated in Section 4. We lastly confront our proposal with related work (Section 5).

2 Context and motivating example

The openess and deperimeterized information system organization involved by intelligent service clouds takes advantage of service reusing abilities to support new business processes. A dynamic supply chain organisation can be seen as a motivating example of such service ecosystem. In this use case a food product tracking process (see Fig. 1) relies on a dynamic workflow interconnecting the business services of the different partners, sharing products production, storage and transport information. Such collaborative workflow combines services and personal workflows from both companies, challenging new security features such as partner authentication, access control on the product storage information, non repudiation features. While setting *on the fly* collaborative organisations, taking advantage of the dynamic service selection provided by the service ecosystem, each partner can compose shared services to create ad-hoc workflows. This involves that a business service can be invoked *on the fly* by different own partner workflows. To provide end to end consistent protection of a given service, one has to take context into account related to the business workflow in which the service takes part and to the end to end execution platform configuration (namely the hosting platform, the access devices and the interconnetion network). Thus, the consistent security services must be composed and orchestrated depending on the execution context. For example, a secured transport is required while accessing logistics information via the unsecured Internet network whereas it is useless while using the safer logistics company LAN, authentication / access control must integrate new partners. Differents methods (EBIOS, MEHARI, OCTAVE, SNA)[3] can be used to identify perceived security risks/system vulnerabilities. Based on the ISO/IEC 27002, the OASIS Service Reference Model[4] defines different security requirements (confidentiality and privacy management, integrity,

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

[4] http://docs.oasis-open.org/soa-rm/v1.0/soa-rm.html

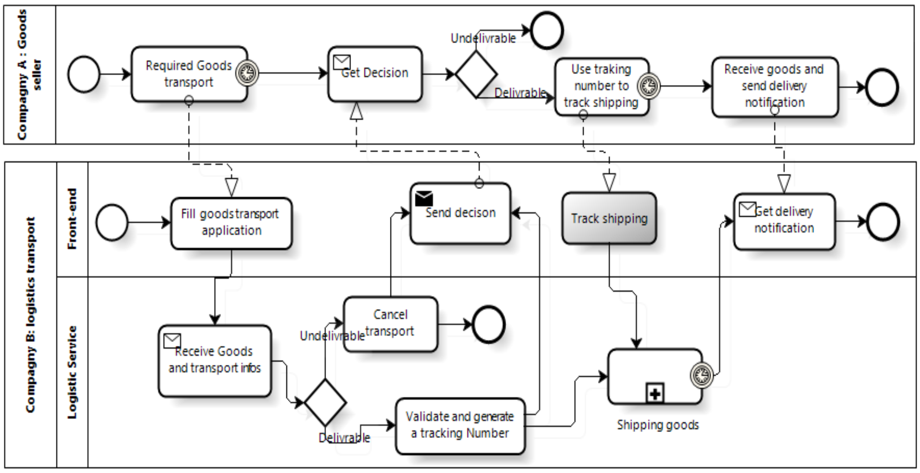
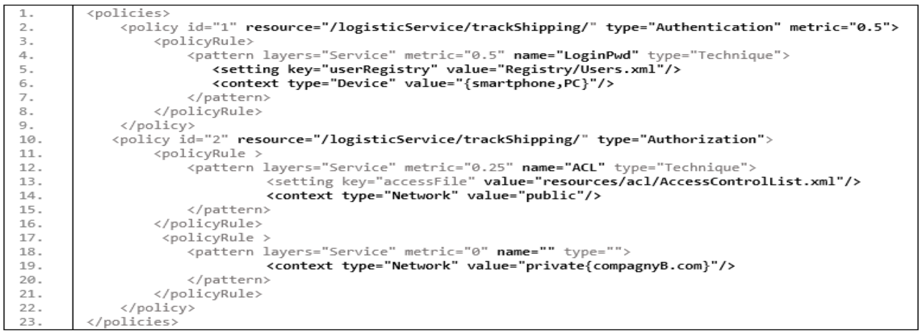

Compagny A : Goods seller
Required Goods transport
Get Decision
Undelivrable
Delivrable
Use traking number to track shipping
Receive goods and send delivery notification
Compagny B: logistics transport
Front-end
Logistic Service
Fill goods transport application
Send decison
Track shipping
Get delivery notification
Receive Goods and transport infos
Undelivrable
Cancel transport
Delivrable
Validate and generate a tracking Number
Shipping goods

Fig. 1. This collaborative business process is organized in different lanes associated to the different supply chain participants. Services such as *Track shipping* are tagged depending on their patrimonial value. Global information on the collaborative environment is also attached to the different lanes. Restricted access control features can also be defined while defining the collaboration agreement between partners.

authentication, authorization, availability and non repudiation) that require the deployment of security means from the network layer, which is rather focused on the availability requirement and protection against deny of service attacks, and transport layer, which has to provide secured confidential channels between transmitters and receivers, to the application layer, which manages most of the security requirements such as authentication, authorization, non repudiation, confidentiality and privacy. Different standards such as WS-Security, SAML, XACML, BSLA, etc. have been developped to support and implement these security requirements. Such protection means can either be deployed directly in the service operation or "attached" to the service interface specification.

Based on the way the OASIS service reference model organizes the different security services, we have proposed in previous work [8] an XML extension to define these security services and protection requirements in the service security policy using exiting standards such as SAML, XACML.

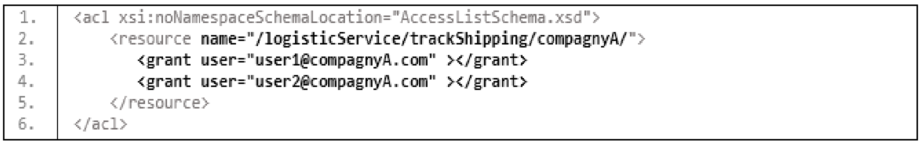
Focusing on the service attached to the logistic application form, a security policy can be set to define the different protection means to be deployed (see Fig. 2). This service includes an operation named TrackShipping, which is considered as a resource (Line 2 in Fig. 2). Its protection requires an authentication (lines 2-9) using a simple login/password process (Line 4) refering to a checking file defined in Line 5. Besides authentication, according to the user network domain (public at Line 14) or company B private network at Line 19) access control rules have to be performed (lines 10-22). If a public network is used, ACL (Line 12), should be applied to this resource. Fig. 3 describes the contents



```
<policies>
    <policy id="1" resource="/logisticService/trackShipping/" type="Authentication" metric="0.5">
        <policyRule>
            <pattern layers="Service" metric="0.5" name="LoginPwd" type="Technique">
               <setting key="userRegistry" value="Registry/Users.xml"/>
               <context type="Device" value="{smartphone,PC}"/>
            </pattern>
        </policyRule>
    </policy>
   <policy id="2" resource="/logisticService/trackShipping/" type="Authorization">
        <policyRule >
            <pattern layers="Service" metric="0.25" name="ACL" type="Technique">
                      <setting key="accessFile" value="resources/acl/AccessControlList.xml"/>
                      <context type="Network" value="public"/>
            </pattern>
        </policyRule>
         <policyRule >
            <pattern layers="Service" metric="0" name="" type="">
                      <context type="Network" value="private{compagnyB.com}"/>
            </pattern>
        </policyRule>
    </policy>
</policies>
```

Fig. 2. Security policies associated to the *Logistic* resource.

of the authorization file allowing only *user1* and *user2* of the company A to access the resource whereas any user of B can access it freely from the B's local network.

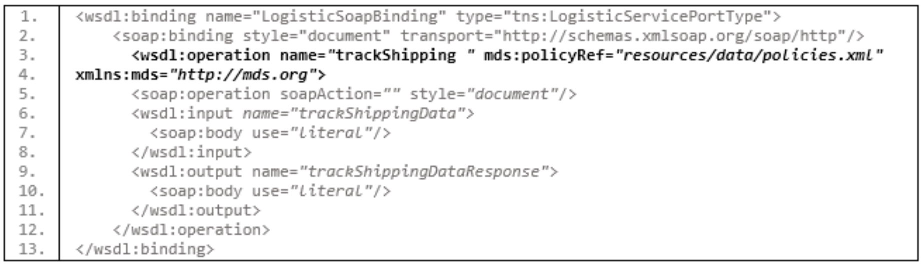


```
<acl xsi:noNamespaceSchemaLocation="AccessListSchema.xsd">
    <resource name="/logisticService/trackShipping/compagnyA/">
        <grant user="user1@compagnyA.com" ></grant>
        <grant user="user2@compagnyA.com" ></grant>
    </resource>
</acl>
```

Fig. 3. *AccessControlList.xml* authorization file.

As these business services can be invoked dynamically by ad-hoc collaborative workflows,or via different devices such as smartphone or classical computing environment, 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.) environment. The protection requirements are defined globally in the security policies attached to the different business services (for example see Fig. 4, 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.

In our example, the tracking service and the related information must be protected in the new *opened* context as it can be accede by partners, with differnt access devices (smartphone for executive members, or PC) . This confidentiality requirement impacts both application layer, which is in charge of the access control, i.e., authentication and authorization management, and transport layer (see Fig. 2). The systematic composition of the authentication and authorization services may be costly. Table 1 shows a comparison of service execution



```
<wsdl:binding name="LogisticSoapBinding" type="tns:LogisticServicePortType">
    <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
      <wsdl:operation name="trackShipping " mds:policyRef="resources/data/policies.xml"
xmlns:mds="http://mds.org">
      <soap:operation soapAction="" style="document"/>
      <wsdl:input name="trackShippingData">
        <soap:body use="literal"/>
      </wsdl:input>
      <wsdl:output name="trackShippingDataResponse">
        <soap:body use="literal"/>
      </wsdl:output>
    </wsdl:operation>
</wsdl:binding>
```

Fig. 4. Link policy file with the *TrackShipping* operation of the *Logistic* service.

time with/without authentication and authorization, testing conditions and environment are detailed in Section 4.

Table 1. Service execution time.

Executed components	Execution time (ms)	
	1-100	101-201
Business service	71	58
Business service + Authentication	82	68
Business service + Authentication + Authorization	85	70

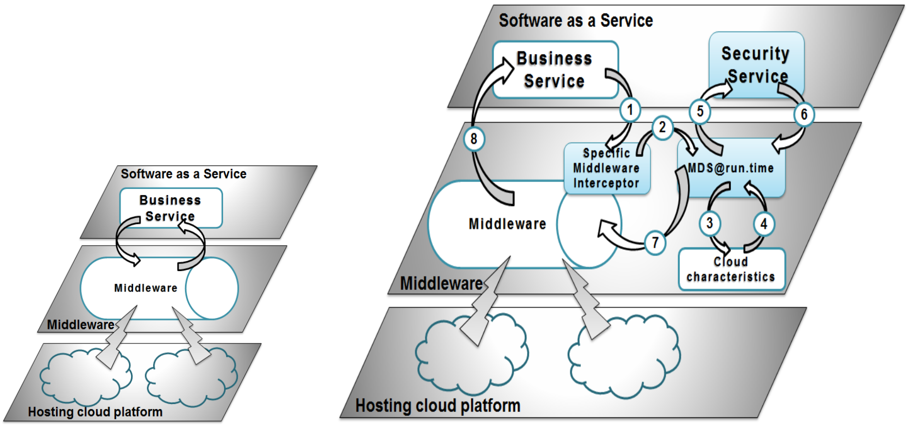
To avoid this costly over protection or risky under protection depending on the runtime environment vulnerability, we propose to turn these security policies as Models@run.time so that they can be analysed to select, compose and orchestrate the most convenient security services depending on the exact 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. In next sections, we present this new architecture and a proof of concept based on the FraSCAti middleware.

3 MDS@run.time with FraSCAti

Our architecture is plugged on the traditional service/middleware/hosting platform architecture (see Fig. 5(a)). In order to avoid under or over protection depending on the runtime context, we propose to outsource the security management thanks to our MDS@run.time architecture (see Fig. 5(b)) which consists in:

- A middleware specific `Interceptor` component plugged on the middleware intercepts each service/middleware interaction (Step 1) and routes this interaction to the `MDS@run.time` component (Step 2).
- The `MDS@run.time` component is the core component to achieve the dynamic security deployment. It consists in three sub components:

  - The `policy manager` parses the service description, extracts and loads the associated policy files before launching the context acquisition process.
  - The `context manager` collects information associated to the execution context (steps 3 and 4). It transfers the results to the `security mediator`.
  - The `security mediator` parses the security policy to get the protection level associated to each security service. Depending on the execution context, it selects and composes security services to implement the required protection. Then it orchestrates the security service invocations (steps 5 and 6) and if succeeded, it routes back the business service/middleware interaction to the middleware (steps 7 and 8).
- The `Security as a Service` component gathers implementation of various security services (authentication, authorization, integrity controls, etc.).



(a) Multi-layer architecture. (b) Multi-layer architecture with MDS@run.time.

Fig. 5. MDS@run.time architecture.

FraSCAti[5] [11] is an open source middleware framework to build, program, deploy, and execute adaptable service-oriented business applications. FraSCAti is based on the OASIS Service Component Architecture (SCA) standard[6]. FraSCAti applications can be deployed in different clouds (Amazon EC2, Amazon Elastic BeanTalk, Google App Engine, CloudBees, etc.)[9][10]. 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 [1]. The adaptability at execution time is based on the FraSCAti reflective features, which encompass introspection and reconfiguration of

[5] `http://frascati.ow2.org`

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

applications at runtime [12]. For dealing with web services and REST, FraSCAti embeds Apache CXF[7], a well-known open source services framework.

To ensure the BP 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) 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. 6.

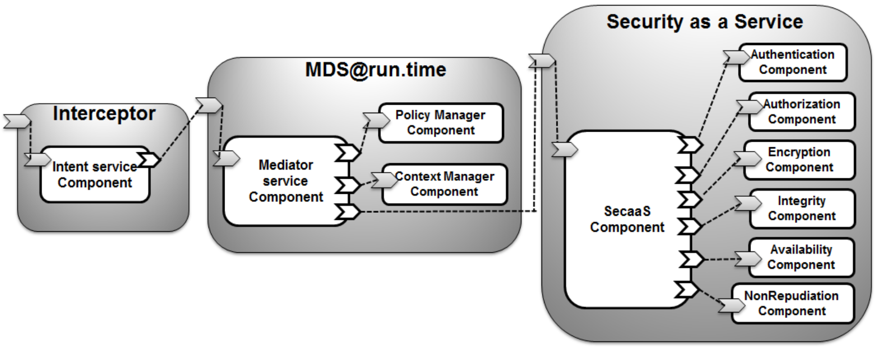


Fig. 6. MDS@run.time with FraSCAti.

3.1 FraSCAti Intent for MDS@run.time

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 components and the underlying service-oriented middleware, allowing on one hand the security services to be able to deploy and run on any

[7] `http://cxf.apache.org`

other middleware and on another hand to deploy on a specific platform just the required security services.

3.2 Composite MDS@run.time

The `MDS@run.time` composite[8] is invoked by the `Intent` component of the `Interceptor` composite. It includes:

– The `Mediator` component is responsible for analyzing called service requests intercepted by the `Intent` component and encapsulated into a `Request` object. It also identifies the security policy rules associated to business services invoked by clients. Thus, through the `Request` object, the `Mediator` receives information of the services involved in the interaction. This information is used to get policies associated to resources (the business services functionality implemented by the service operation). These policies are then analyzed and orchestrated by the `Mediator` to call the required security services.
– The `PolicyManager` component manages the 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.

3.3 Composite SecaaS

The `Security as a Service` (SecaaS) composite is invoked by the `Mediator` component. It includes various security services, which allow protecting resources and business services according to a *security as a service* approach. This composite contains the following components:

– The `SecaaS` component is the composite entry point. It receives from the `Mediator` component the security policies to be applied. It is responsible for analyzing these policies, to identify the type of security services (authentication, authorization, etc.) to call.
– 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`

[8] An SCA composite is an SCA component containing a set of SCA components.

component includes subcomponents such as `SSORegistry` (Single Sign On Registry) component used to store information about authentication of sessions and to allow to retrieve user information without restarting authentication.

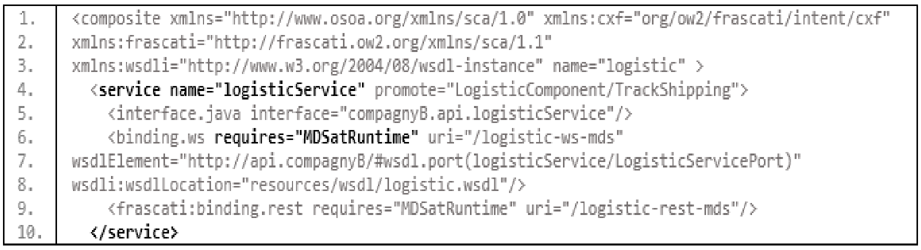
- The `Authorization` component allows managing access to resources and services, and allows grant or deny the user access to them. As the `Authentication` component, it receives the security policy rule and invokes the authorization mechanism to be applied. This mechanism can be based on an authorization 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 and secure communication protocols (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) 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.

4 Evaluation

Our performance evaluation is based on the use case presented in Fig. 1, focusing on the *TrackShipping* operation. This operation is implemented thanks to a service associated to a security policy including authentication (see Fig. 2, lines 2-8) by login/password (Line 4) and access control (lines 9-16) using ACL (Line 11) combined with a used network constraint (Line 13). As far as the collaborative service is concerned, the business service is encapsulated in an *LogisticService*, which is associated to the convenient security policy and refers to the `MDS@run.time` composite (Fig. 7, Line 6). By this way, the business service can be intercepted and MDS@run.time is invoked before invoking the business service itself.

To evaluate the impact of our *MDS@run.time with FraSCAti* prototype on the service execution time, we set a test 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. We set different types of measures: A first execution invokes the business service without invoking our security architecture (measure 1 used to set a reference time), the time between the intent invocation and the MDS@run.time



```
<composite xmlns="http://www.osoa.org/xmlns/sca/1.0" xmlns:cxf="org/ow2/frascati/intent/cxf"
xmlns:frascati="http://frascati.ow2.org/xmlns/sca/1.1"
xmlns:wsdli="http://www.w3.org/2004/08/wsdl-instance" name="logistic" >
  <service name="logisticService" promote="LogisticComponent/TrackShipping">
    <interface.java interface="compagnyB.api.logisticService"/>
    <binding.ws requires="MDSatRuntime" uri="/logistic-ws-mds"
wsdlElement="http://api.compagnyB/#wsdl.port(logisticService/LogisticServicePort)"
wsdli:wsdlLocation="resources/wsdl/logistic.wsdl"/>
    <frascati:binding.rest requires="MDSatRuntime" uri="/logistic-rest-mds"/>
  </service>
```

Fig. 7. Link the *Logistic* component with **MDS@run.time**.

invocation measures the cost for the service interception. Then, the time required to get the policy file and parsed it is expressed by the mediation measure. Lastly execution times related to the authentication process and to the authorization process are given. We manage a test loop to compute an average time for the first request to evaluate the setup time and on 200 client requests, so that extra factor impacts can be smoothed, such as bootstrapping effects, Just-In-Time compilation, etc.

Table 2. Mean execution time of **MDS@run.time** components.

No	Component	Average execution time (ms) time (ms)		Average execution time/ Total execution time	
		1-100	101-201	1-100	101-201
1	FraSCAti + Apache CXF + Business service	63	53	74%	75%
2	FraSCAti Interceptor	1	1	2%	2%
3	MDS@run.time	7	4	8%	6%
4	Authentication	11	10	13%	14%
5	Authorization	3	2	4%	3%
	Total	85	70	100%	100%

The main result is that the interception and mediation process represents only 8% (101-201 requests) of the total execution. This overhead could certainly reduced within an industrial implementation of **MDS@run.time**. However this demonstrates that our **MDS@run.time** approach, i.e. interpretation of security policies at runtime, introduces a small overhead compared to over protecting services.

5 Related work

Different strategies can be used to provide a consistent protection on distributed information systems, paying attention on both organisational and infrastructure related risks.

On one hand, *Security by Design* approaches integrate protection requirements while designing the information system. To this end, different frameworks have been defined to manage security annotations on UML diagrams (such as the multi-purpose UMLSec or the rather access control oriented Secure UML domain specific languages) or BPMN diagrams [13]. Taking advantage of such high-level specification, MDS [6] adapts the Model Driven Engineering (MDE) approach to the security field. Several studies have focused on the use of the MDS approach to secure BP and led to frameworks definition like OpenPMF [5], and SECTET [2]. Nevertheless, none of them support the full transformation process. While BPSec is focused on the requirement engineering part (it includes CIM and PIM models), SECTET and Open PMF provide PIM, PSM and code generation features. Moreover, the generation process is achieved according to a static environment vision (perimetrised information system and well-known deployment platform). This does not fit the dynamic service ecosystem context.

On the other hand, the security stack defined in the OASIS Service reference model defines the way protection requirements should be implemented in a multi-layer architecture (application/Middleware-Transport/network) to improve the global protection consistency while deploying security features associated to a standardized security policy. Nevertheless, this coarse-grain model does not integrate any platform dependent risks / protection models (such as works achieved for cloud based infrastructure by the Cloud Security Alliance (CSA)[9], Intrusion Detection System [7], vulnerabilities checking [3], etc.) nor any governance loop so that the execution context can be taken into account while deploying the required protection. This can lead to either over or under protection. To overcome this limit, our MDS@run.time approach takes advantage of the OASIS security model and of the MDS approach to generate security policies depending on the collaborative BP organisational context so that services can be secured on demand. Moreover, it provides a fully oursourced security environment that can be plugged on any service-oriented middleware. 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.

6 Conclusion

Securing collaborative business processes deployed on cloud systems requires 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 FraSCAti middleware and

[9] `http://www.cloudsecurityalliance.org/guidance/csaguide.v3.0.pdf`

evaluate its performance level. 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. Acher, M., Cleve, A., Collet, P., Merle, P., Duchien, L., Lahire, P.: Reverse Engineering Architectural Feature Models. In: Springer (ed.) 5th European Conference of Software Architecture (ECSA). In Computer Science, vol. 6983, pp. 220–235. Springer, Essen, Allemagne (Sep 2011), `http://hal.inria.fr/inria-00614984`
2. Alam, M., Hafner, M., Breu, R.: Constraint based role based access control in the SECTET-framework A model-driven approach. Journal of Computer Security pp. 223–260 (2008)
3. Avgerinos, T., Cha, S.K., Rebert, A., Schwartz, E.J., Woo, M., Brumley, D.: Automatic Exploit Generation. Commun. ACM 57(2), 74–84 (Feb 2014), `http://doi.acm.org/10.1145/2560217.2560219`
4. Blair, G., Bencomo, N., France, R.B.: Models@run.time. Computer 42(10), 22–27 (2009)
5. Lang, U.: OpenPMF SCaaS: Authorization as a Service for Cloud & SOA Applications. In: Cloud Computing Technology and Science (CloudCom), 2010 IEEE Second International Conference on. pp. 634–643 (Nov 2010)
6. Lcio, L., Zhang, Q., Nguyen, P.H., Amrani, M., Klein, J., Vangheluwe, H., Traon, Y.L.: Chapter 3 - Advances in Model-Driven Security. In: Memon, A. (ed.) Advances in Computers, vol. 93, pp. 103 – 152. Elsevier (2014), `http://www.sciencedirect.com/science/article/pii/B9780128001622000038`
7. Modi, C., Patel, D., Borisanya, B., Patel, A., Rajarajan, M.: A Novel Framework for Intrusion Detection in Cloud. In: Proceedings of the Fifth International Conference on Security of Information and Networks. pp. 67–74. SIN '12, ACM, New York, NY, USA (2012), `http://doi.acm.org/10.1145/2388576.2388585`
8. Ouedraogo, W.F., Biennier, F., Ghodous, P.: Adaptive Security Policy Model to Deploy Business Process in Cloud Infrastructure. In: 2nd International Conference on Cloud Computing and Services Science (CLOSER 2012). pp. 287–290 (2012)
9. Paraiso, F., Haderer, N., Merle, P., Rouvoy, R., Seinturier, L.: A Federated Multi-Cloud PaaS Infrastructure. In: 5th International Conference on Cloud Computing (CLOUD'12). pp. 392–399. IEEE (2012)
10. Paraiso, F., Merle, P., Seinturier, L.: soCloud: A service-oriented component-based PaaS for managing portability, provisioning, elasticity and high availability across multiple clouds. Special Issue on Cloud Computing, Computing Journal, Springer (To appear 2014)
11. Seinturier, L., Merle, P., Fournier, D., Dolet, N., Schiavoni, V., Stefani, J.B.: Reconfigurable SCA applications with the FraSCAti Platform. In: IEEE International Conference on Services Computing (SCC'09). pp. 268–275. IEEE (2009)
12. Seinturier, L., Merle, P., Rouvoy, R., Romero, D., Schiavoni, V., Stefani, J.B.: A component-based middleware platform for reconfigurable service-oriented architectures. Software: Practice and Experience 42(5), 559–583 (2012)
13. Wolter, C., Menzel, M., Schaad, A., Miseldine, P., Meinel, C.: Model-driven business process security requirement specification. Journal of Systems Architecture (JSA) pp. 211–223 (2009)