ANR programme ARPEGE 2008

Systèmes embarqués et Grandes Infrastructures

Projet SELKIS : Une méthode de développement de systèmes d'information médicaux sécurisés : de l'analyse des besoins à l'implémentation.

ANR-08-SEGI-018

Février 2009 - Décembre 2011

Implementation of Web Services for Security Enforcement

Livrable numéro 4.2

Gouenou Coatrieux
Télécom Bretagne - LaTIM Inserm U650
Michel Embe Jiague
LACL
Stéphane Morucci
SWID

February 2011

Abbreviations

AOP Aspect Oriented Programming

ASTD Algebraic State Transition Diagram

BPEL Business Process Execution Language

ESB Enterprise Service Bus

HTTPS Hypertext Transfer Protocol Secure

IdP Identity Provider

IS Information System

JAX-WS Java API for XML-Based Web Services

PDP Policy Decision Point

PEM Policy Enforcement manager

PEP Policy Enforcement Point

RBAC Role Based Access Control

SAML Security Assertion Markup Language

SAML2 Security Assertion Markup Language 2

SOA Service Oriented Architecture

SOAP Simple Object Access Protocol

SP Service Provider

SSO Single Sign On

WS Web Service

WSDL Web Services Description Language

XACML eXtensible Access Control Markup Language

XML eXtensible Markup Language

XSD XML Schema Document

Contents

List of Figures

1 Context

In the SELKIS project, security enforcement is achieved using Web Services (WSs) that are customized by a trusted external process. In this document, we detail identified WS, their implementation and how they are orchestrated inside Med.e.com and Ifremont softwares. We also show how policies are deployed to security components. An alternative approach to security enforcement in Information System (IS) is presented in Section 4. This approach relies on a formal language to express security policies and an automated mechanism that derives an enforcement framework for Service Oriented Architecture (SOA) applications.

We are relying on a proven concept, namely delegating security mechanisms to dedicated and specific entities. We are going further by making it possible to manage these security entities using concrete policies that meet high level requirements, thanks to an advanced derivation mechanism.

In this document, some common security requirements are implicit.

Confidentiality:
To ensure confidentiality properties, all communications between components rely on Hypertext Transfer Protocol Secure (HTTPS). If necessary, we can release some constraints by requiring that only communications with the user (and therefor, communications from an untrusted zone) be made over HTTPS.

Integrity:
To ensure integrity properties, all messages between components rely on digital signatures. Alternatively, we could require that only communications from/to an untrusted zone are signed.

Availability:
All components of the system are protected enough to avoid denial of services.

Authenticity:
All components in this system have enough information to mutually authenticate. Trusted keys have already been distributed to all components of the system.

Non-repudiation:
All information are logged, signed and encrypted to establish authenticity and non-repudiation.

2 Medecom

2.1 Delegated Authentication

2.1.1 Security Policies

The security policies that applies here are quite simple and do not need to be derived using a complex mechanism from a high level security requirement. This security policy just states that a user must be authenticated to access Medecom resources. The user authenticates using his login and his password.

2.1.2 Architecture

Figure 1 details how authentication is delegated inside Medecom Web application.

The User makes an initial request to the Medecom application (the Service Provider (SP)) and is redirected to an authentication server (the Identity Provider (IdP)) using the secured Security

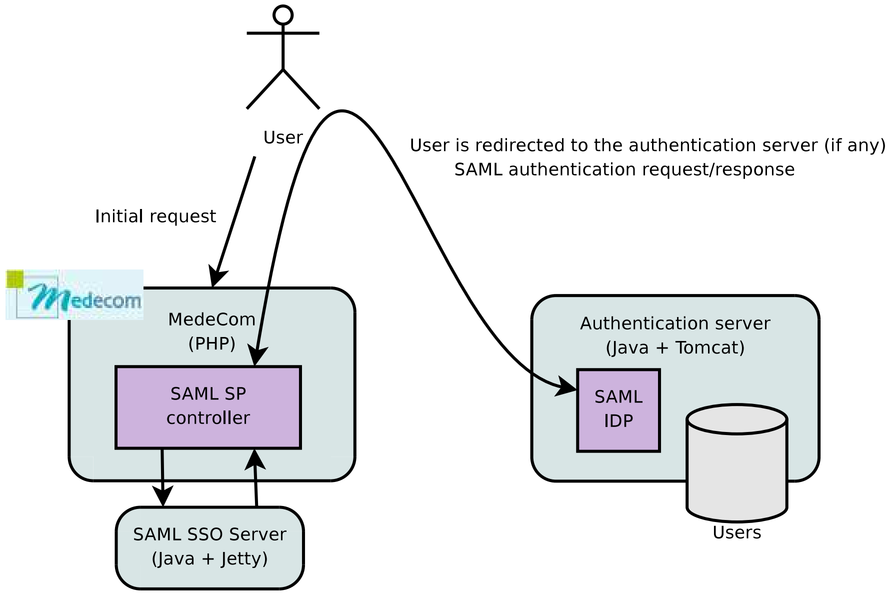

Figure 1: Authentication delegation for Medecom

Assertion Markup Language 2 (SAML2) protocol. The User is then authenticated on a dedicated server and redirected with the necessary credentials back to the Medecom software.

For this mechanism to occur properly, we introduced two specific servers and a PHP client library to be integrated inside Medecom's Web application.

- The PHP client library is in charge of receiving (non-authenticated) incoming requests, generating SAML2 Authentication Requests and parsing Security Assertion Markup Language (SAML) Authentication Responses. In fact, this library is a proxy that forwards requests and responses to the SAML Single Sign On (SSO) Server, which parses and generates SAML messages.
- The second server is the authentication server (Auth Server) which is in charge of receiving and parsing SAML2 Authentication Requests, and generating SAML2 Responses.

2.1.3 Workflow

Figure 2 depicts the UML representation of the corresponding workflow.

This simple workflow is hard-coded inside Medecom/SAML SP softwares. There is no need of a workflow engine to manage this process.

2.1.4 Web Services

The following WS have been implemented and are available for deployment.

2.1.4.a WS located at the SAML SSO Server

The SAML SSO server offers two WS (see Table 1). It also offers two others WS for authorization management that are detailed later in this document.

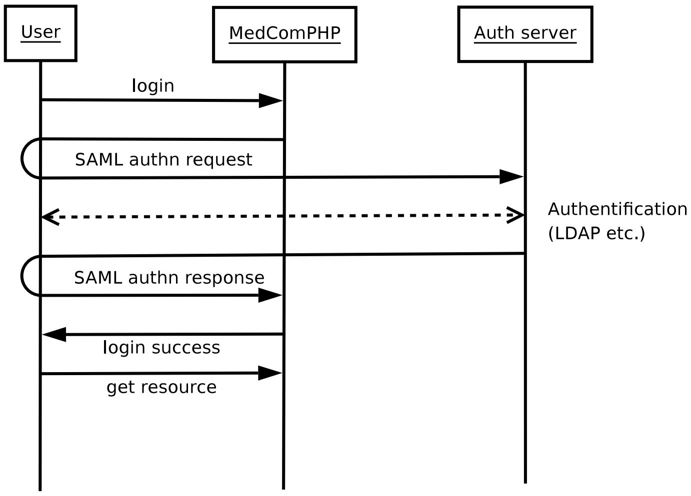

Figure 2: Authentication delegation workflow for Medecom

Table 1: WS located at the SAML SSO Server

Function	**Input**	**Output**	**Description**
Generate SAML SSO request	Method: GET /?resource=<resource name> or Method: GET /	SAML2 signed authentication request.	Use case: The User wants to access a resource identified by its name. The User is not yet authenticated. The SP needs to send a SAML2 Authentication request to the IdP. WS purpose: this WS generates a signed SAML request on behalf of the SP. The SP forwards this request to the IdP. The resource parameter is optional but if provided, the WS creates a mapping between the SAML request identifier and the resource the user wants to access.
Handle SAML SSO response	Method: POST /?action=sso_response &SAMLResponse= <response>	Key-value form response. Parameters: status: ok/error entityID: < the SamlIdpServer that authenticated the user> nameID:<an identifier that represents the user> resource:<resource>	Use case: the SP received a SAML SSO response from a SamlIdpServer and it does not know how to handle it. WS purpose: this WS receives and parses a SAML2 Authentication Response (generated by the IdP). The SP parses this key-value form to determine if the user is authenticated or not. If a mapping exists between a resource and the initial SAML request ID, then the key-value form response value is this resource. For instance, it will be used by the SP in order to redirect the user to this resource.

The SAML SSO Server offers a specific protocol, simple enough to facilitate implementations for multiple technologies (Ruby, PHP, CGI, etc...). It is then possible to have a simple library at the client side using simple WS to communicate to a Java server that handles all the necessary SAML2 mechanisms.

2.1.4.b WS located at the IdP Server

In this context, this WS receives SOAP+SAML authentication requests from a SP (see Table 2). It relies on a user database to authenticate users and returns signed SAML assertions to the SP.

It also handles other types of requests that are detailed later.

Table 2: WS located at the IdP Server

Function	Input	Output	Description
Handle SAML SSO request	Method: GET or POST /do/samlsso? SAMLRequest= <request>	HTML form for the user to authenticate.	Use Case: An anonymous user is redirected to the SamlIdpServer with a base 64 encoded SAML Authentication request. WS purpose: Parses and interprets SAML2 Authentication requests. The SAML IdP Server initiates a user session to handle SSO. In case of user/password authentication, this WS generates a form that is displayed to the user. Since this WS implies user interaction, it may not be considered as a real WS.
Handle SAML SSO request	Method: GET or POST /do/samlsso? SAMLRequest= <request> Session identifier from IdP Server inside cookie.	SAML2 authentication response.	Use Case: An authenticated user is redirected to the SamlIdpServer with a base 64 encoded SAML Authentication request. SamlIdPServer authenticates the user and redirects the user to the SP with a base 64 encoded SAML response. This WS is the same as above. It differs in the sense that the incoming request also contains credentials related to a previous successful authentication. In this case, an HTML form is not necessary (SSO mechanism).
Authenticate user	Method: POST Parameters: - Login - Password (In session: calling service, I.e. SP)	SAML2 authentication response.	WS purpose: this WS is in charge of authenticating the user. The IdP server holds all the necessary information to check credentials provided by the user. Upon successful authentication, the user is redirected to calling SP using a SAML2 authentication response.
Handle SAML single logout protocol	Method: GET or POST /do/singlelogout	SAML2 Logout request	Use Case: Logout of a user from multiple services. This use case is the exact opposite of the SSO mechanism. WS purpose: Parses and interprets a SAML logout request. The SAML IdP Server invalidates the user session. This WS generates a SAML2 Logout response.

2.2 Delegated Authorization

2.2.1 Security Policies

We used the security policy document from Medecom in order to define a policy expressed in the OrBAC model with organizations, roles, activities, views, contexts entities. We defined two organizations "Server" and "Medecom_Server". The first one defines abstract entities, hierarchies and rules, the second one affects concrete entities from the Medecom application (in PHP) to abstract entities. The organization "Medecom_Server" is a sub-organization of "Server" and inherits all rules, entities etc. previously defined.

Our authorization server reads eXtensible Access Control Markup Language (XACML) policies, the MotOrBAC tool can be used in order to write the policy and then XACML policies are generated. It is also possible to write the OrBAC policy in a text format and then to translate it into XACML policies (or into a RDF file which can be read by MotOrBAC).

(Cf annexes)

This policy does not define subjects empowered into roles because the Medecom Application handles these affectations. New users or roles can be created and this policy does not need to be updated. There is a limited number of actions and objects (URL resources) in the application, and they are affected to either views or activities. For instance the "users_profiles" object is used in the "Administration" view.

Figure 3 shows the role hierarchy defined in the policy designed using the MotorBac Flex application. We can see there is a default role, every user empowered into a role will be also empowered into this default role. Then we distinguish two roles "Unauthenticated_User" and "Utilisateur" which is a default role for any authenticated user of the application.

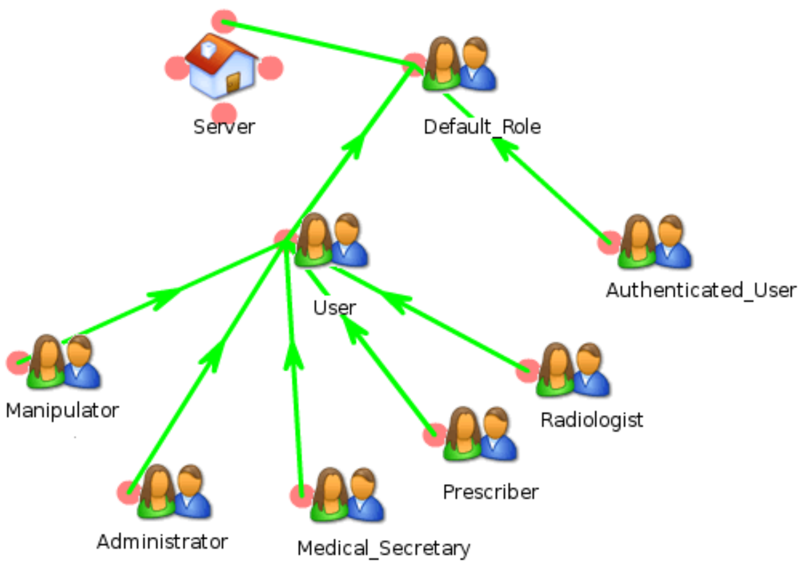

Figure 3: Role hierarchy

2.2.2 Architecture

Figure 4 details how authorization is delegated inside Medecom Web application when an authenticated user wants to access a resource.

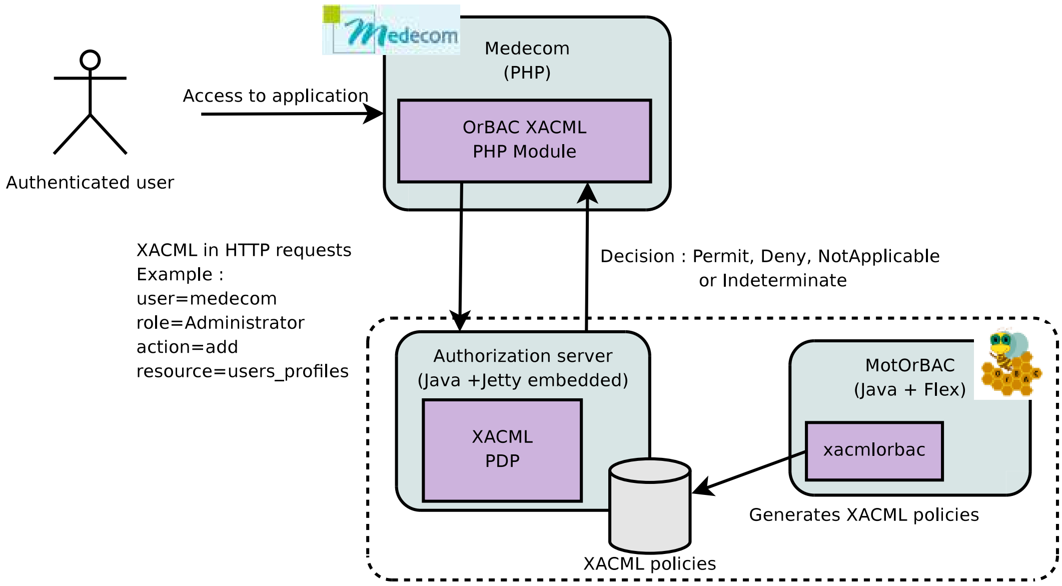

Figure 4: Authorization delegation in Medecom

Medecom policies are defined using the MotOrBAC tool and corresponding XACML policies are generated. The authorization server will evaluate requests against these policies. In the Medecom application resources are web pages such as a profile page or list of studies, actions can be adding, editing etc. Each time a user wants to perform an action on a resource, the OrBAC XACML PHP module sends an XACML request to the authorization server which contains the user, role, action and resource attributes. For instance the user can be "medecom" with the role "Administrateur" wanting to perform the action "add" on the resource "user_profiles".

From the security policy, "add" is considered as the activity "Ajouter" which is sub-activity of "Administrer". The resource "user_profiles" is used in the view "Administration". The permission "administrateur_p0" applies and then access is permit. The authorization server returns its decision "Permit" to the OrBAC XACML PHP module and the application enforces this decision: the user is allowed to add a new profile.

If a prohibition rule applies then the decision is "Deny" and the application refuses the user action. When the authorization server dodn't find any rule to apply (either permission or prohibition) then its decision is "NotApplicable". In a closed policy a "NotApplicable" decision is considered as a "Deny" decision by the Medecom Application. The last decision result is "Indeterminate" which occur when the authorization server encounters an error such as conflicts between permissions and prohibitions if the policy has not been well-defined.

2.2.3 Workflow

This simple workflow is hard-coded inside Medecom/SAML SP softwares without using a workflow engine to manage this process. When the user wants to perform an action on a resource, the Medecom application sends an XACML request to the authorization server. The latter evaluates the request against the XACML policies and returns its decision to the Medecom application. The XACML policies used by the authorization server are generated from the MotOrBAC tool.

2.2.4 Web Services

The following WS have been implemented and are ready for deployment. The Medecom application already handles authorization (access to pages, role affectation, etc.). Whereas this is coded in the PHP application, and it was possible to access some administration pages without being an administrator.

When we delegated access control to our authorization server, it fixed this security flaw, each time a user wants to perform an action (such as read) a request is sent to the authorization server. Moreover when authorization is delegated, if the policy changes we only have to edit the policy at the authorization server. There is no need to edit the PHP code of the application.

2.2.4.a WS located at the SAML SSO Server

In this context, the SAML SSO server delegates authorization to an authorization server, and returns the decision it received from it. There is another WS located at the SAML SSO server in order to handle the SAML name id mapping when the authorization server wants to share an identifier with the IdP which authenticated the user. All those WS are described in Table 3.

Table 3: WS located at the SAML SSO Server

Function	**Input**	**Output**	**Description**
Evaluate authorization request	Method: POST /?action=authz &nameID=<user> &providerID= <provider> &resourceID= <resource> &organization= <organization>	Key-value form response parameters. decision:Permit decision:Deny decision:NotApplicable decision:Indeterminate	Use case: SP delegates authorization and sends an authorization request to the authorization server. WS purpose: This WS generates an XACML request and delegates the authorization to a SAML Authorization server. It will get a XACML decision. This WS will generate a key-value response with this decision parameter.
Handle SAML name id mapping request	Method: POST /?action= nameid_mapping &SAMLRequest= <request>	SAML response that contains a new encrypted NameID identifier.	Use case: When evaluating an XACML request, the authorization server may want some attributes about the user but it did not share a identifier with the IdP. So the authorization server asks the SAML SSO server a new identifier about this user only shared with the IdP. WS purpose: This WS already shares an identifier about this user with the IdP. It will send a SAML name id mapping request to IdP and will receive a new identifier, encrypted with the public Policy Decision Point (PDP) key. So the SAML SSO server cannot read it, this identifier will be only shared between SAML IdP and the authorization server.

2.2.4.b WS located at the IdP Server

During the authorization workflow, two WS are available at the IdP server. One WS can handle SAML name id mapping requests from the SP in order to share a common identifier with the authorization. The other WS can handle SAML attribute requests from the authorization server. All those WS are described in Table 4.

Table 4: WS located at the IdP Server

Function	Input	Output	Description
Handle SAML attribute request	Method: POST /do/-queryAttributes	SAML response with attributes (such as role).	Use case: Some attributes may be necessary when the authorization server evaluates an XACML request. It will send a SAML attribute request to the SAML IdP server. WS purpose: This WS returns attributes about the NameID identifier in the request.
Handle SAML name id mapping request	Method: GET or POST /do/singlelogout	SAML response with a new encrypted NameID identifier.	Use case: A SP shares an identifier about this user with the IdP. And the SP shares an identifier about this user with the authorization server. But no identifier about this user is shared between the IdP server and authorization server. So SP asks a identifier representing this user that will be shared between the IdP server and the PDP server. WS purpose: This WS generates a new identifier representing the user. This identifier is encrypted using the authorization public key. And a SAML response containing this new identifier is returned to the SP.

2.2.4.c WS located at the Authorization Server

We propose four WS located at the authorization server. The main WS can evaluate XACML request. The three other WS are not implemented yet, they will be used by the administration part, such as updating the policy or setting contexts. All those WS are described in Table 5.

Table 5: WS located at the Authorization Server

Function	Input	Output	Description
Evaluate request	Method: GET or POST /<policy name>/?req=<request>	Key-value form response parameters. decision:Permit decision:Deny decision:NotApplicable decision:Indeterminate	Use case: SP delegates authorization and sends an XACML request to the authorization server. WS purpose: The XACML base 64 encoded request is evaluated against the policy <policy name> (if existing). This WS generates a key-value form response with a decision parameter.
Update/Deploy policy	(not yet implemented)	(not yet implemented)	Use case: MotOrBAC tool edits a policy and it wants to deploy or update the policy on the authorization server.
Ecosystem API	(not yet implemented)	(not yet implemented)	Use case: an emergency context in a hospital is enabled. Some rules will apply with this context. WS purpose: contexts are enabled or disabled, the authorization server decision will depend on these contexts when evaluating XACML requests.
Access logs	(not yet implemented)	(not yet implemented)	Use case: Audit system needs the access logs generated by the authorization server when it evaluated requests.

2.2.4.d WS located at the SAML Authorization Server

We also propose five WS located at the SAML authorization server. The main WS can evaluate XACML request. The three other WS are not implemented yet, they will be used by the administration part, such as updating the policy or setting contexts. The difference with the previous authorization server is that this server handles SAML requests. XACML requests are embedded into SOAP+SAML requests and can be digitally signed and encrypted. All those WS are described in Table 6.

Table 6: WS located at the SAML Authorization Server

Function	Input	Output	Description
Evaluate signed request	Method: POST /<policy name>/	XACML+SAML+SOAP Message is signed	Use case: SP delegates authorization and sends a SOAP+SAML+XACML request to the authorization server in the POST body content. Request is signed. WS purpose: The XACML request is evaluated against the policy <policy name> (if provided). This WS generates an XACML decision embedded in a SAML+SOAP message.
Evaluate encrypted and signed request	Method: POST /<policy name>/encrypted	XACML+SAML+SOAP Message is signed and encrypted	Use case: SP delegates authorization and sends a SOAP+SAML+XACML request to the authorization server in the POST body content. Request is signed and encrypted. WS purpose: The XACML request is evaluated against the policy <policy name> (if provided). This WS generates an XACML decision embedded in a SAML+SOAP message.
Update/ Deploy policy	(not yet implemented)	(not yet implemented)	Use case: MotOrBAC tool edits a policy and it wants to deploy or update the policy on the authorization server.
Ecosystem API	(not yet implemented)	(not yet implemented)	Use case: an emergency context in a hospital is enabled. Some rules will apply with this context. WS purpose: contexts are enabled or disabled, the authorization server decision will depend on these contexts when evaluating XACML requests.
Access logs	(not yet implemented)	(not yet implemented)	Use case: Audit system needs the access logs generated by the authorization server when it evaluated requests.

2.3 Image Integrity and Authenticity Control using Watermarking Technologies

For medical image content protection, we propose a set of services based on the lossless watermarking of some security attributes into the images. These services, called by the image applications in connection or not with WS allows verifying the image integrity and its origin (authenticity). The access to the watermark and image integrity/authenticity checking is conducted by a watermarking module. Services request depends on the knowledge of a secret key (Ks), a public key (Kpu) and a private key (Kpr). The watermark verification relies on ks and Kpu, and its update on Ks and Kpr, Kpu. The first implementation of our watermarking algorithm has not been yet integrated into the telemedicine platforms.

Table 7: Image Integrity and Authenticity Control using Watermarking Technologies

Function	**Input**	**Output**	**Description**
Image protection	Image to be protected, service requested (integrity, authenticity or both), image unique identifier and keys (Ks, Kpr,Kpu),	The watermarked image, success embedding flag, message really embedded.	Use case: The image application requests the watermarking module to protect an image. It provides its keys and the protection service requested (integrity, authenticity or both of them). If authenticity protection is requested, it should give the image UID. Service purpose: the image is watermarked and protected in terms of integrity, authenticity or both.
Image protection verification	Image to be controlled, requested service (checking of integrity, authenticity or both), image unique identifier and keys, access to the unwatermarked image	The unwatermarked image, success reading flag, embedded message, value of the image integrity and authenticity.	Use case: The image application requests the watermarking module to verify the image integrity, or authenticity or both of them. If authenticity verification is requested, it should give the image UID for the checking. It can also require the unwatermarked image. Service purpose: the watermarked image is protected in terms of integrity, authenticity or both.
Image protection update	Image to be re-protected, requested service to update (checking of integrity, authenticity or both), image unique identifier and keys.	The watermarked image, update success flag, message really embedded.	Use case: The image application requests the watermarking module to update the image protection. It provides its keys and the protection service requested to be updated (integrity, authenticity of both). If authenticity update protection is requested, it should give the image UID. Service purpose: the image is re-watermarked or the watermark content is updated as in the case of an image transmission.

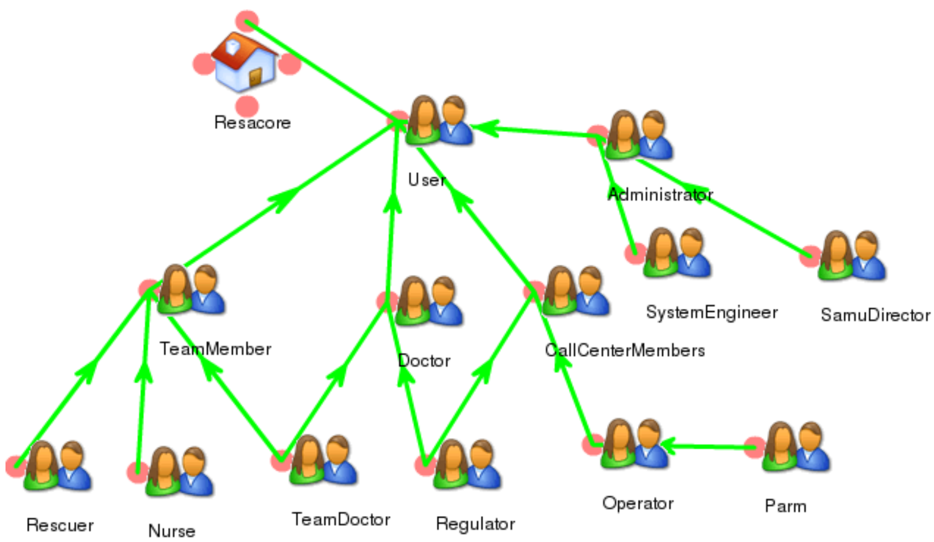

Figure 5: Role hiverarchy using MotOrBAC Flex application

3 Ifremmont Implementation

3.1 Delegated Authorization

3.1.1 Security Policies

(cf Annexes)

In this policy, rules are defined for the roles "User", "Regulator", "Parm" and "TeamMember". The default role has no rights on the management acts. A "Regulator" is a doctor at the call center, he can give medical advices, instructions, prescriptions or diagnostics to patients. A "Parm" can only gives medical advices to patients. A "TeamMember" is a prehospital actor who is sent to the patient, he can access medical information about the patient if he participates to the intervention.

Figure 5 shows the role hierarchies defined in the policy using the MotOrBAC Flex application. The default role is "User" and other roles inherit from it.

3.1.2 Architecture

Figure 6 details how authorization is delegated inside Res@mu application when an authenticated user wants to access a resource. Security policies are defined with MotOrBAC, then XACML policies are generated and are used by the authorization server.

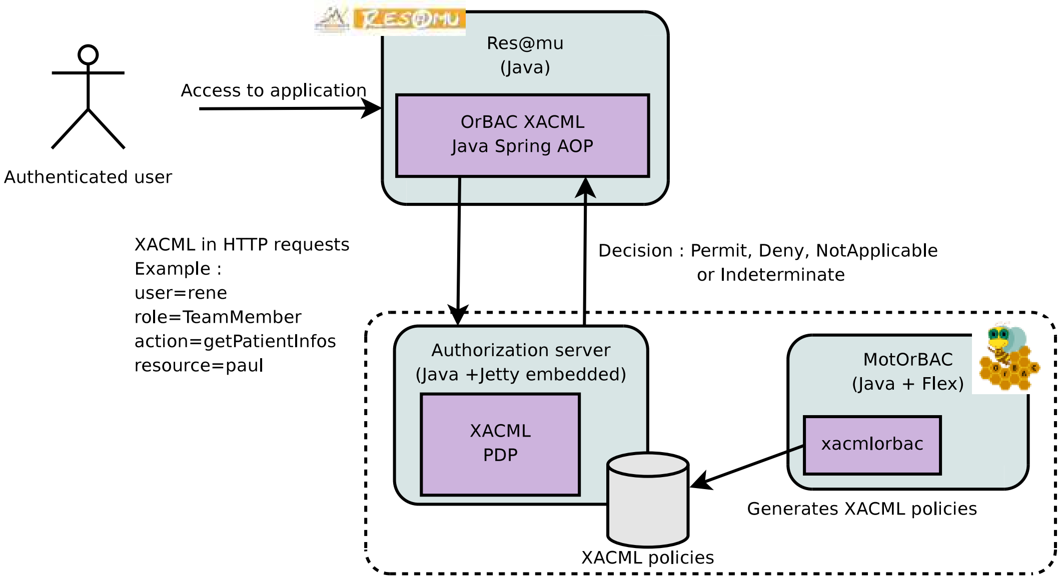

Figure 6: Authorization delegation for Res@mu

We use the Java Spring AOP framework — AOP stands for Aspect Oriented Programming — in order to delegate access control when requesting Res@mu services. For instance, if a user calls the "getPatientInfos" method on a "Patient" object, then an XACML request is sent to the authorization server. The Res@mu application handles the user authentication and role affectation such as "TeamMember". The authorization server evaluates the request against the XACML policies generated with MotOrBAC and returns a decision. The latter can be either "Permit", "Deny", "NotApplicable" or "Indeterminate".

3.1.3 Workflow

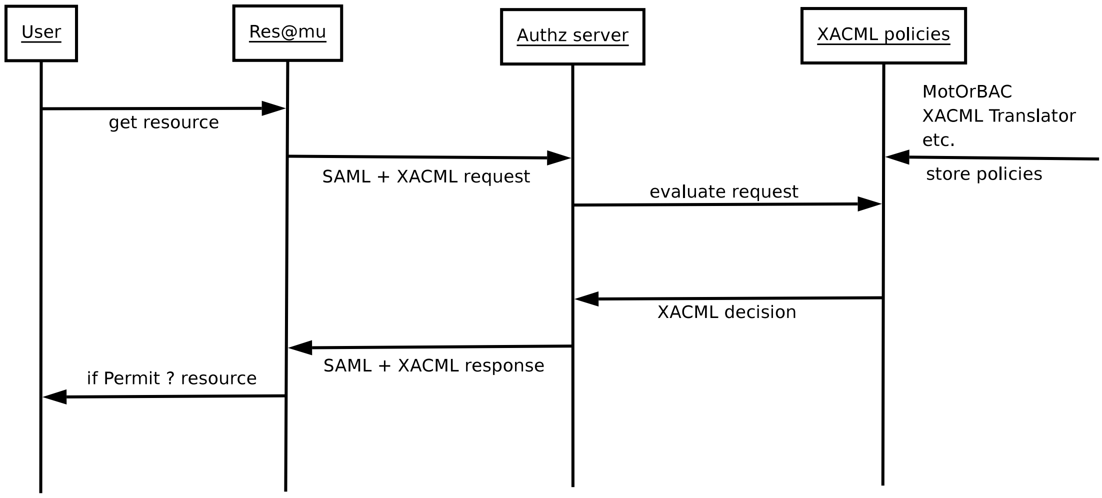

Figure 7: Authorization delegation workflow for Res@mu

This simple workflow depicted in Figure 7 is hard-coded inside Ifremmont Res@mu software. There is no need of a workflow engine to manage this process. Using AOP, access control is injected in the service layer, and a XACML request embedded in each SAML message sent to the authorization server. Only the "Permit" decision allows granting access to resources.

3.1.4 Web Services

The following WS have been implemented and are available for deployment.

3.1.4.a WS located at the Authorization Server

We propose four WS located at the authorization server (see Table 8). The main WS can evaluate XACML requests. The three other WS are not implemented yet, they will be used by the administration part, such as updating the policy or setting contexts.

3.1.4.b WS located at the SAML Authorization Server

We also propose five WS located at the SAML authorization server (see Table 9). The main WS can evaluate XACML request. The three other WS are not implemented yet, they will be used by the administration part, such as updating the policy or setting contexts. The difference with the previous authorization server is that this server handles SAML requests. XACML requests are embedded into SOAP+SAML requests and can be signed and encrypted.

Table 8: WS located at the Authorization Server

Function	Input	Output	Description
Evaluate request	Method: GET or POST /<policy name>/?req= <request>	Key-value form response parameters. decision:Permit decision:Deny decision:NotApplicable decision:Indeterminate	Use case: SP delegates authorization and sends an XACML request to the authorization server. WS purpose: The XACML base 64 encoded request is evaluated against the policy <policy name> (if provided). This WS generates a key-value form response with a decision parameter.
Update/ Deploy policy	(not yet implemented)	(not yet implemented)	Use case: MotOrBAC tool edits a policy and it wants to deploy or update the policy on the authorization server.
Ecosystem API	(not yet implemented)	(not yet implemented)	Use case: an emergency context in a hospital is enabled. Some rules will apply with this context. WS purpose: contexts are enabled or disabled, the authorization server decision will depend on these contexts when evaluating XACML requests.
Access logs	(not yet implemented)	(not yet implemented)	Use case: Audit system needs the access logs generated by the authorization server when it evaluated requests.

Table 9: WS located at the SAML authorization server

Function	Input	Output	Description
Evaluate signed request	Method: POST /<policy name>/	XACML+SAML+SOAP Message is signed	Use case: SP delegates authorization and sends a SOAP+SAML+XACML request to the authorization server in the POST body content. Request is signed. WS purpose: The XACML request is evaluated against the policy <policy name> (if provided). This WS generates an XACML decision embedded in a SAML+SOAP message.
Evaluate encrypted and signed request	Method: POST /<policy name>/encrypted	XACML+SAML+SOAP Message is signed and encrypted	Use case: SP delegates authorization and sends a SOAP+SAML+XACML request to the authorization server in the POST body content. Request is signed and encrypted. WS purpose: The XACML request is evaluated against the policy <policy name> (if provided). This WS generates an XACML decision embedded in a SAML+SOAP message.
Update/ Deploy policy	(not yet implemented)	(not yet implemented)	Use case: MotOrBAC tool edits a policy and it wants to deploy or update the policy on the authorization server.
Ecosystem API	(not yet implemented)	(not yet implemented)	Use case: an emergency context in a hospital is enabled. Some rules will apply with this context. WS purpose: contexts are enabled or disabled, the authorization server decision will depend on these contexts when evaluating XACML requests.
Access logs	(not yet implemented)	(not yet implemented)	Use case: Audit system needs the access logs generated by the authorization server when it evaluated requests.

4 A PEM Implementation for ASTD Security Policies

Another rigorous approach to policy enforcement is being explored through the use of a formal language to express security policies and automatic translation procedures to derive enforcement frameworks. The approach focuses on the authorization part of security/policy enforcement. The overall architecture is presented in Figure 8. The main components of the enforcement framework are the Policy Enforcement Point (PEP) and the PDP. The PEP interacts with the WS clients, the IS itself as a more or less coordinated set of WS and the PDP to enforce authorization decisions inferred by the PDP from the security policy. The implementation of both components are described in the two following sections.

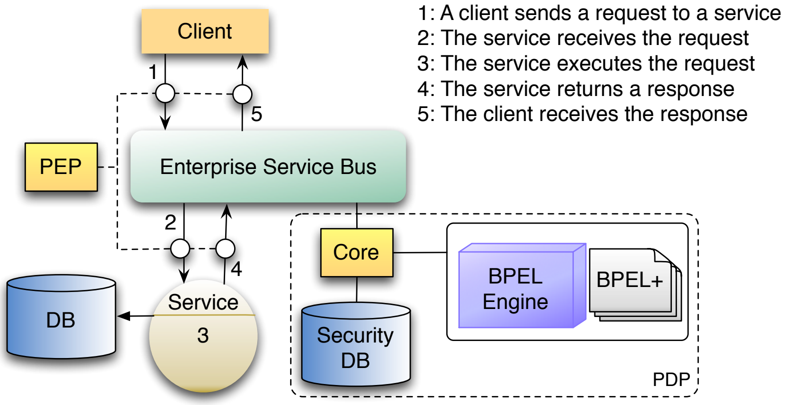

Figure 8: Enforcing a policy in a SOA environment

4.1 Implementation of the PEP

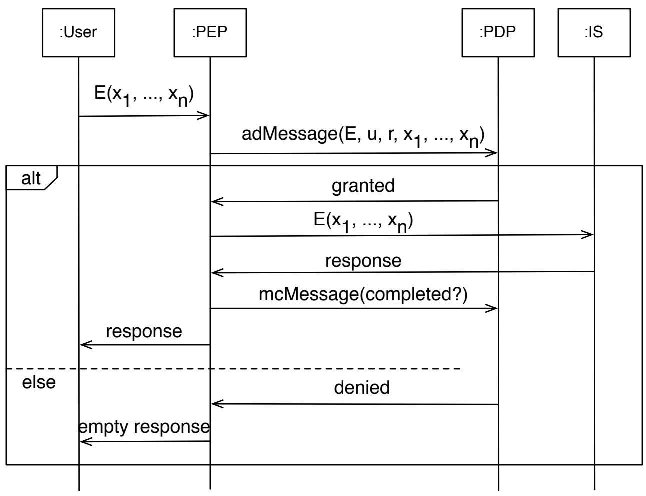

Figure 9: PEP sequence diagram

Figure 9 depicts the sequence of messages between the application's user and the PEP on one hand and the PEP and the PDP on the other hand. When the user request a service $E(x_1, ..., x_n)$ from the IS, the request is intercepted by the PEP. The latter then proceed by sending an authorization's request to the PDP with the message $adMessage(E, u, r, x_1, ..., x_n)$. The authorization request contains the requested service name E, the user identity u, his acting role r and the service parameters x_1 to x_n since they may be required by the security policy. In the use case considered in our prototype implementation, only the user identity and its roles are of interest in the regards of the security policy. The PDP may respond to the authorization's request with authorization decision *granted* or *denied*. In the first case, the PEP will let the initial user request flow to the IS and which response will be transmitted back to the user through the PEP. In the second case (the authorization is *denied*), the initial request is not executed by the IS.

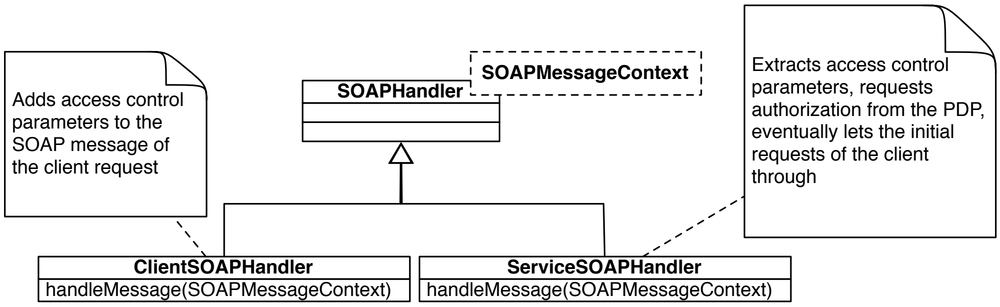

Figure 10: SOAP handlers

The PEP is implemented by two WS handlers. Since the implementation of the IS relies Java API for XML-Based Web Services (JAX-WS)[1], those handlers are plain Simple Object Access Protocol (SOAP) handlers placed between the WS client and the Enterprise Service Bus (ESB) on one hand and the ESB and the WS itself on the other hand. Figure 10 describes the handlers hierarchy implemented in our prototype.

4.2 Implementation of the PDP

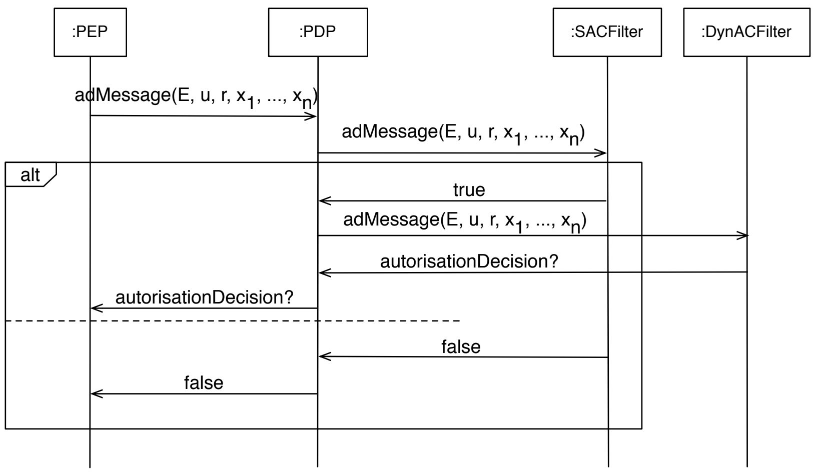

Figure 11: PDP sequence diagram

Figure 11 depicts the sequence of messages between the PEP and the PDP on one hand, and the PDP et its filters on the other and. Our implementation relies on two filters in order to render an authorization decision:

- a *static filter*, implemented as a set of RBAC compliant database relations and the corresponding Java code to query those relations;

[1] http://jcp.org/en/jsr/detail?id=224

- a *dynamic filter*, implemented by a BPEL process exposed as a WS.

The current PDP implementation uses both filters and synthesizes an authorization decision through a conjunction of the response from the two filters. The scenario in Figure 11 shows a dynamic filter that returns a negative response. The dynamic filter implemented as a BPEL process is derived from a formal Algebraic State Transition Diagram (ASTD) specification.

In [1], a two-step translation procedure is described. From an ASTD security policy specification with building blocks derived from patterns, the procedure creates a BPEL filtering process and the required WSDL and XSD documents since the process is deployed on the ESB as a WS. In another approach [2], the dynamic filter is implemented by a BPEL containing a lightweight Javascript interpreter for ASTD specifications. Also embedded within the process, is the security policy specification encoded as an XML variable and provided as the first input of the lightweight Javascript. Figure 12 describes the input as well as the output of the Javascript interpreter.

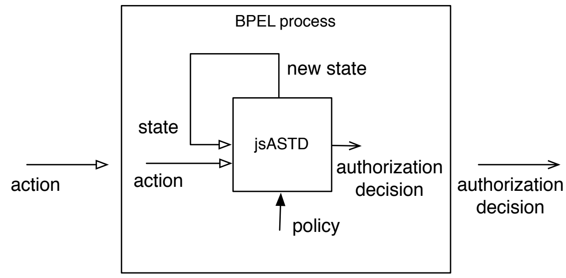

Figure 12: jsASTD, a lightweight interpreter in a BPEL process

5 Annexes

5.1 Policies

5.1.1 Medecom

5.1.1.a Roles

Table 10: Roles

Role	Description	Tasks
Radiologist	Radiologist in a town or in the radiology service of an hospital.	Performs the ultrasound examinations Writes and validates reports Diagnoses
Manipulator	Medical electroradiology manipulator.	Welcomes patients in the exam room Get images Prints images
Medical secretary	Medical secretary is responsible for welcoming patients and prints medical records on the information system of HIS (Hospital Information System).	Welcomes patient Prints smedical reports Makes appointments
Prescriber	General practitioner or specialist	Analyses results Completes medical reports
Administrator	Administrator is responsible for system information	Software management User management Hardware management

5.1.1.b Resources

Table 11: Resources

Resource	Description
Exams/series (worklist)	Table with the list of patients exams.
Profiles	Profiles define access rights based on roles.
Users	User information such as attributes, password, username.
User settings	An user can modify a subset of his attributes (password, email, etc).
Cache and SMTP	Cache is a folder that contains all the images.
Log files and audits	Log files collect information about server processes. Audit centralizes information about user authentication, access to an examination etc.
Statistics	Statistics such as how many online users.
User registration	Register form for a new user.

5.1.1.c Actions

Table 12: Actions

Action	Description
Create/Add	The user creates or add a resource.
Read	The user reads a resource.
Modify	The user modifies a resource.
Delete	The user deletes a resource.
Search	The user searches a resource using filters.
Email notification	The user triggers a notification for a prescriber when a new examination is available.
Register	The user fills the register form.
Export	The user exports a resource to a PDF file.
Download	The user downloads a resource on his computer.

5.1.1.d Security Policy

Table 13: Security policy

Resource	Role	Action	Context
Exams/series (worklist)	Radiologist Manipulator Administrator	Read Search Delete Email notification	
	Medical secretary	Read Search Email notification	
	Prescriber	Read Search	Only his patients
Profiles Users	Administrator	Read Add Delete Modify Search	
User settings	Radiologist Manipulator Medical secretary Prescriber Administrator	Read Modify	
Cache and SMTP	Administrator	Read Modify	
Log files and audits	Administrator	Read Search	
Statistics	Radiologist Administrator	Read	
User registration	Prescriber	Add/Create	

5.1.2 Res@mu

5.1.2.a Roles

Table 14: Roles

Role	Description	Tasks
User	Generic role for any user accessing the Res@mu system.	Generic user, no task
Administrator	Administrator is responsible for system information.	
SystemEngineer	Responsible for the technical tasks of the software.	
SamuDirector	Responsible for actions of his teams. Only performs administrative tasks in order to satisfy legal or medical obligations.	
CallCenterMember	Person who picks up the phone when someone calls the SAMU and may perform management acts.	
Operator	The first person who picks up the phone, this person has no medical formation and is responsible for collecting patient's information. This person has no access to medical records.	
PARM	Operator who has received a medical formation. This person can give medical advices.	Gives medical advices
Doctor	Doctors can perform management acts. They can be a regulator or a team doctor.	
Regulator	After an Operator or PARM collects informations about the patient, the call is assigned to a Regulator (Doctor). The latter can send a team to the patient.	Accesses medical records Performs management acts
TeamMember	Rescue TeamMember sent to the patient.	Access to medical records Perform management acts
Rescuer	Rescuer is a TeamMember with a minimal medical formation.	
Nurse	Ambulance technician with a medical formation, can perform management acts under the doctor responsibility.	
TeamDoctor	Doctor in a team sent to the patient. He is responsible for the management acts performed by his team.	

5.1.2.b Resources

Table 15: Resources

Resource	Description
PatientView	List of management acts on patients.
ManagementAct	A management act such as medical advice, prescription or instruction.
Patient	Medical record and information about the patient.

5.1.2.c Actions

Table 16: Actions

Action	Description
Get patient information	Get medical record and management acts of a patient.
Validate	Validate a management act.
Add/Create	Create a management act.
Modify	Modify a management act.

5.1.2.d Security Policy

Table 17: Security policy

Resource	Role	Action	Context
Patient view	TeamMember	Create ManagementAct	Participate in the intervention
	PARM	Add valid ManagementAct	Participate in the intervention and management act is an instruction or prescription or a medical advice.
	Regulator	Add valid ManagementAct	Participate in the intervention and management act is a medical advice.
Management act	User	No access (prohibition)	
	PARM	Read	Only read medical advices.
	Regulator	Read	
	TeamMember	Validate	A team member can only validate his management acts.
		Read	Participate in the intervention.
		Modify	ManagementAct has not been not validated.
Patient	TeamMember	Read	Participate in the intervention.

References

[1] M. Embe Jiague, M. Frappier, F. Gervais, R. Laleau, and R. St-Denis. From ASTD access control policies to WS-BPEL processes deployed in a SOA environment. In *The 1st International Symposium on Web Intelligent Systems & Services, WISS 2010*, volume to be published, 2010.

[2] M. Embe Jiague, F. Gervais, R. Laleau, and R. St-Denis. A BPEL Implementation of a Security Filter. In *PhD Symposium at the 8th IEEE European Conference on Web Services*, 2010.