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

Functionalities of the Policy Enforcement Manager

Livrable numéro 4.1

Michel Embe Jiague
LACL
Stéphane Morucci
Swid

Février 2010

Abbreviations

AJAX Asynchronous JavaScript and XML

BPEL Business Process Execution Language

CIM Computation Independent Model

DBMS Database Management System

EB^3 Entity-Based Black-Box

EB^{3}SEC EB^{3}Secured

ESB Enterprise Service Bus

HTTPS Hypertext Transfer Protocol Secure

IS Information System

JAX-WS Java API for XML-Based Web Services

JSP JavaServer Pages

LDAP Lightweight Directory Access Protocol

OASIS Organization for the Advancement of Structured Information Standards

OrBAC Organization-Based Access Control

PAP Policy Administration Point

PDP Policy Decision Point

PEM Policy Enforcement Manager

PEP Policy Enforcement Point

PIP Policy Information Point

RBAC Role Based Access Control

SAML2 Security Assertion Markup Language 2

SELKIS SEcure health care networKs Information Systems

SOA Service Oriented Architecture

SOAP Simple Object Access Protocol

WSDL Web Service Description Language

WSS Web Services Security

XACML eXtensible Access Control Markup Language

XML eXtensible Markup Language

Contents

List of Figures

1 Introduction

In the WP4 of the SEcure health care networKs Information Systems (SELKIS) project proposal, Information Systems (ISs) are implemented using Web services (in the broad sense). Security features are implemented in the Policy Enforcement Manager (PEM). Those services rely on data available in relational databases or eXtensible Markup Language (XML) based files. Our security framework is based on two mains actors: the Policy Enforcement Point (PEP) and the Policy Decision Point (PDP). They are responsible of intercepting client application's requests to services and applying security policies on those requests. There are two other actors to consider: the Policy Administration Point (PAP) which allows to manage the security policies in a policy repository and the Policy Information Point (PIP) which provides additional information on request's subjects (roles, actions/services, environment, ...) when required by the PDP.

1.1 The Policy Enforcement Point component

The PEP intercepts all requests from client applications/components to each distributed component of the overall IS. If the IS's architecture has a single entry point for such request (most unlikely in todays Service Oriented Architecture (SOA) deployments), then it might be the perfect candidate where to implement a PEP.

At the transport level, communication with the PEP must benefits from the state of the art technology to ensure *confidentiality* and *integrity*. Such technologies may encompass cryptography and digital signatures. For this purpose, XML Enc and XML Sig are examples of such technology that may provide confidentiality and integrity features respectively for Simple Object Access Protocol (SOAP) based Web services. More generally, Organization for the Advancement of Structured Information Standards (OASIS) proposes the standards Web Services Security (WSS) for Web services which provide the required features to ensure confidentially and integrity.

1.2 The Policy Decision Point component

The PDP is responsible of making approval/denial decisions based on defined policies in the policy repository. The decision is based on three different levels of functional security as depicted in figure 1 to ensure a high degree of fine tuned confidentiality.

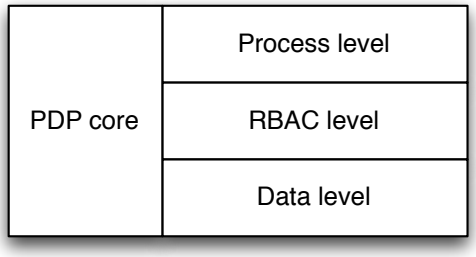

Figure 1: PDP internal view

1.2.1 Decision at the database level

The lowest level is at the database and "elementary" access to data. At this level, access to elementary functions (read, write, delete, ...) are checked and filters on accessed data (on both read and write operations) are applied. For example, a doctor may be able to access the records for his patients only. Depending on the designer of the application, a request can specify the access to patient records and limits the result to the records associated with the doctor. In this case the application logic (access patient's records) is mixed with a security rule. But the implementation may also specify the request to access patient's records, without enforcing at the same time any restriction on the returned results, and leave the latter to the PDP. Implementations might harness existing Role Based Access Control (RBAC) functionalities and constraints capabilities already existing in most Database Management System (DBMS).

1.2.2 Decision at the second level using OrBAC

The second level is the service level. Actions and operations of the IS are elements to secure i.e. to control access to. Their granularity depends on the policy designer, so they can be high-level services or even lower purpose services. Security policies at this level are RBAC in essence. PDP must enact those policies. SWID is working on an engine enacting eXtensible Access Control Markup Language (XACML) representations of those policies.

In the next paragraphes, we are focusing on a policy enforcement manager driven by security policies expressed in Organization-Based Access Control (OrBAC) [1]. All these policies are input and validated in terms of correctness and consistency using the freely available MotOrBAC tool, and more specifically using its inference engine. This design corresponds to the Computation Independent Model (CIM) level described in SELKIS project.

The PEM must rely on a PAP/PDP and PEP common architecture. The PEP handles user requests. The PDP transforms a high-level security policy expressed in OrBAC into configurations applicable to a PEP (or answer specific PEP-generated requests). The PAP represents MotOrBAC and its high-level security policy.

This PEM must also integrate an existing architecture, namely some Med.e.com software components already installed. The overall picture is depicted in figure 2.

Motorbac[1]
This tool manages high-level security policy and ensures they are complete and correct.

Authentication server (Authent. server)
The authentication server is in charge of authenticating users and delivering credentials using a standardized protocol.

Authorization server (Authz. server)
Once authenticated, users are given rights to access Med.e.com resources. This is handled by the authorization server which is also using a standardized protocol.

[1] http://motorbac.sourceforge.net/index.php?page=home&lang=fr

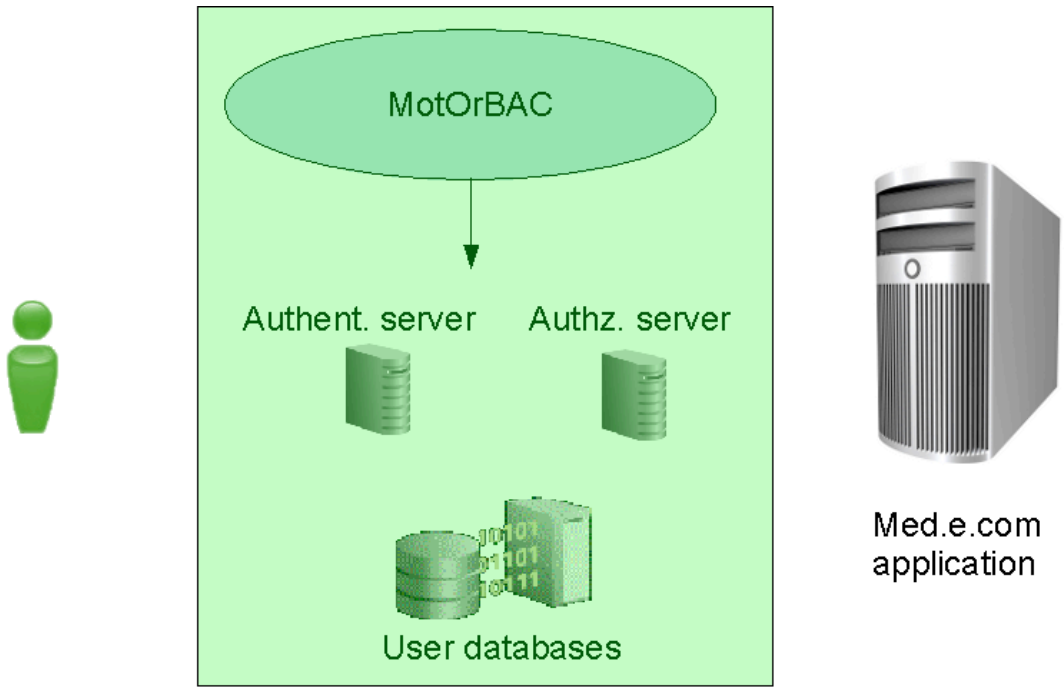

Figure 2: Overall picture

Using established standards guarantees that the authentication and/or the authorization could be easily replaced in the infrastructure if Med.e.com or its customer should change providers. It is also a mean to be independent from security software vendors by providing some "plug-and-play" components.

User databases
These databases contain the information required for the authentication server and the authorization server to run properly.

Finally the Med.e.com application needs to be modified to integrate the output of the authentication and the authorization servers. In particular:

- authenticated users must be recognized as such by the Med.e.com application
- Med.e.com application must be modified to take into account granted access (or denied access) to resources.

As mentioned above, policies are expressed using OrBAC. Since we need to integrate into an existing system that may already have some security or usage policies specified in another language, it is interesting to define and introduce an intermediate simple formalism that could be translated from OrBAC (with the same level of expressiveness) and that could also be produced by internally developed tools to leverage existing policies mastered by hospital managers. Indeed, converting existing policies to a XACML rule set is a complex and error-prone task. Using a more user-friendly intermediate language will lead to better integration.

Commercial products may generate XACML policies. They can also be customized to generate policies expressed in such an intermediate language using for instance a two-steps process:

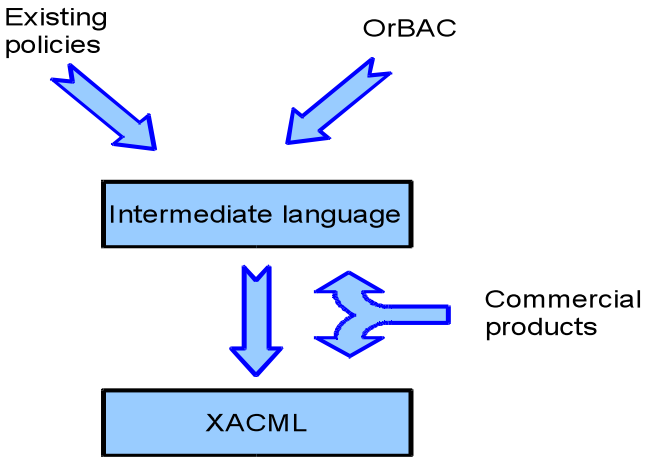

Figure 3: From an OrBAC rule to a XACML policy

first an export from existing policies and then a dedicated transformation process between the commercial product proprietary output format and this intermediate language.

This intermediate language, which still needs to be defined, must be in text format to facilitate its generation by some external process or software. The more input format the PEM can handle, the better.

1.2.3 Decision at the process level using BPEL

The third level is the process level. At this level, the PDP makes decisions on executing actions in the context of processes. The decision engine is based on a Business Process Execution Language (BPEL) engine enacting policy rules as BPEL processes [5]. As such, the rules are not attached to the actions or services to secure, nor to the entities (roles, actors, ...) involved. Furthermore, those rules are expressed in a formal language [3] and then, are automatically transformed to BPEL processes (see figure 4). For example, billing a patient is done only after the patient have receive care from a doctor or a nurse, so a rule may specify that the service performing the billing operation can be invoked only after a care for the patient has been registered.

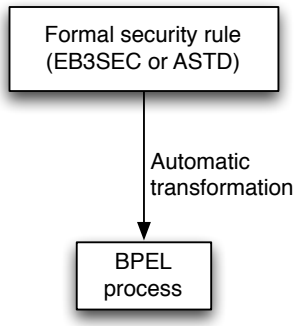

Figure 4: Transformation from a formal language to BPEL

BPEL is an XML-based language standardized by OASIS. It is best suited for "long lasting" processes in organizations. It can be used to orchestrate workflows. Futhermore, it is very well integrate with Web services, since a BPEL process interface with (caller and callee) partners is also described using Web Service Description Language (WSDL)[2]. Even if BPEL is XML-based, some editors propose graphical environments for BPEL documents, such a Netbeans[3] (see figure 5).

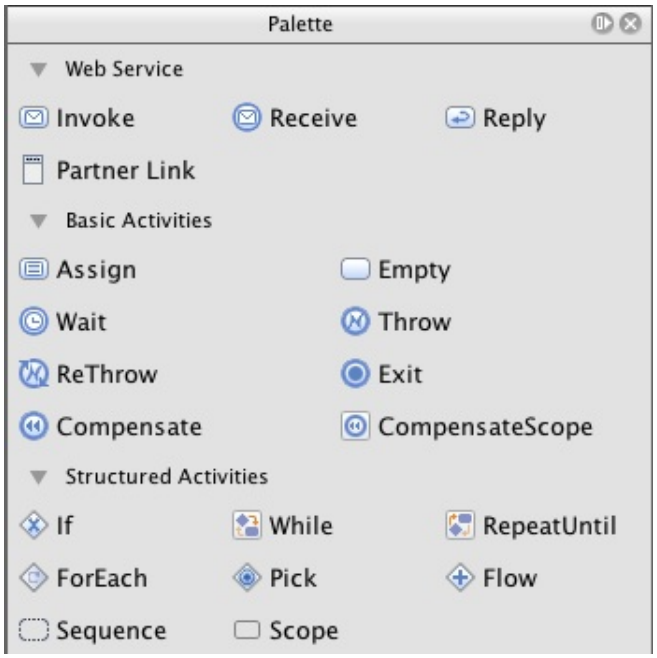

Figure 5: BPEL activities in the Netbeans toolbox

1.3 The Policy Administration Point component

Policies must be specified by different stakeholders who can have different privileges or management rights. The tool used to manage associated policies must take into account these constraints by displaying an interface automatically adapted to the administrator profile (i.e the administrator management rights). Moreover, some actions that are not permitted for a given administrator must not be displayed for this administrator. This is mainly to simplify the interface, to make it coherent and to avoid confusion among users. By showing only what is necessary, we are facilitating the acceptance of this new tool.

Using OrBAC, it is possible to define the administration rights. It is not yet possible to automatically adapt the interface according to this administration policy.

Since administrators are potentially located anywhere in an hospital, it is crucial to have a centralized server hosting all policies. To provide seamless integration, a Web-based interface to manage these centralized policies appears as a good choice. Users only need their day-to-day browser to administrate policies they are in charge of. This is simple and flexible and represents a de-facto standard. Today state of art also requires an interface dynamically built by the

[2] http://www.w3.org/TR/wsdl
[3] http://netbeans.org/

client, i.e a so-called "Web 2.0" interface, using technologies like Asynchronous JavaScript and XML (AJAX), Silverlight or Flash/Flex.

Such objectives are interesting challenges that need to be addressed.

2 Scenarii of the PEM

The following paragraphs describe the two main scenarii that take place in the PEM.

In scenario (a) in figure 6, the user application sends a request (1) to a service (or a component of the distributed application) along with some credentials (e.g. user identification, role, etc.) on a secure channel insuring confidentiality and integrity of the exchanged message. The request is intercepted by the PEP which extracts the credentials and then formulate an autorisation request for approval/denial by the PDP (2). The PDP makes a decision on whether or not to approve or deny the client application's request (in scenario (a) the request is denied). The denial is then sent back to the PEP (3) which transmits it to the initial client application (4).

In scenario (b), messages (1) and (2) are the same as in the scenario (a) but in this case the client application's request is approved. The autorisation is reported back (3) to the PEP. The PEP then allows the original request to continue to requested service/component (4). The last steps are the service responding to the user application's request (5) and (6). The service may also take steps to perform – other business – validation on the request (e.g. checking that an account has enough credit before performing a fund's transfer). This response goes through the PEP so that security policy repository or PAP (if there is any) can be updated with the recently executed request.

It is important in both cases (a) and (b) that all message exchanges must be carry out through secure channels. Those communication schemes are simplifications of the security data-flow diagram described in the XACML's OASIS standard [4].

3 Deployment of the PEM in a SOA environment

3.1 Building a SOA environment from scratch

In a typical SOA environment, the services are deployed as Web services "behind" an Enterprise Service Bus (ESB) serving as a gate to route messages, among other features, from consumers to the components of the environment (see figure 7).

The PEP is implemented as a Web service *handler* on both the consumer and the service side. On the consumer side, the handler role is to inject security parameters (credentials made off user identity, role, ...) in the SOAP request message. On the server side, the handler is in charge of retrieving the security parameters from the SOAP request and asking the PDP for a decision on the access request using the given security parameters. This implementation is not intrusive, however it requires that the consumer application implements a protocol in the cases where the request is denied. This can be done using the exception mechanism or using error codes.

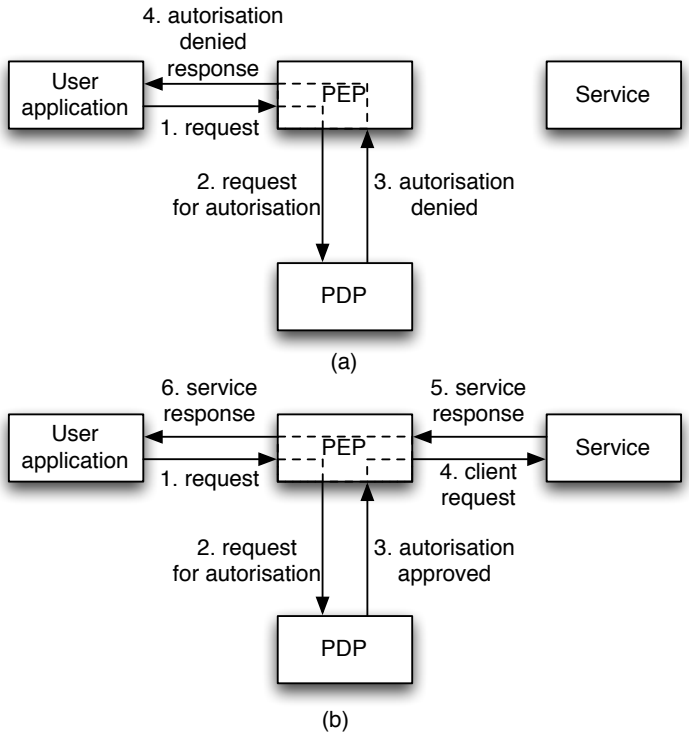

Figure 6: PEP and PDP security scheme

3.1.1 Prototype

In [3], Konopacki et al. expose an example of a system illustrating a deposit of a cheque in a bank. The example is specified using EB[3] a formal method for specifying IS [2]. The system is made off three entities as shown in figure 8: an instance of the class `deposit` is created to link a `client` and a `cheque`.

However, we have implemented the IS as a set of Web services as depicted in figure 9 and deployed in Glassfish[4] them according to the architecture in figure 7. The Web services maintain the entities as database records in the light DBMS Derby[5].

Along with those Web services, we have implemented a client application (consumer in figure 7) as a JSP page and a servlet. The page does not implement the common usage flow of Web application i.e. a user first log in and then he can use the services offered by the application. It is a rather simple Web page where the credentials are input as the same time as the parameters (the new value of a cheque to modify for example) of the service to use.

As mentioned earlier, the PEP is based on two handlers, each on one side of the ESB. Both are implemented as sub-classes of the Java API for XML-Based Web Services (JAX-WS)[6] framework's generic interface `SOAPHandler` (see figure 10). These lies down to the implementation of the method `handleMessage(SOAPMessageContext)`. The handler on the consumer side is responsible to inject security parameters in SOAP request originating from the consumer. This can

[4] https://glassfish.dev.java.net/
[5] http://db.apache.org/derby/index.html
[6] http://jcp.org/en/jsr/detail?id=224

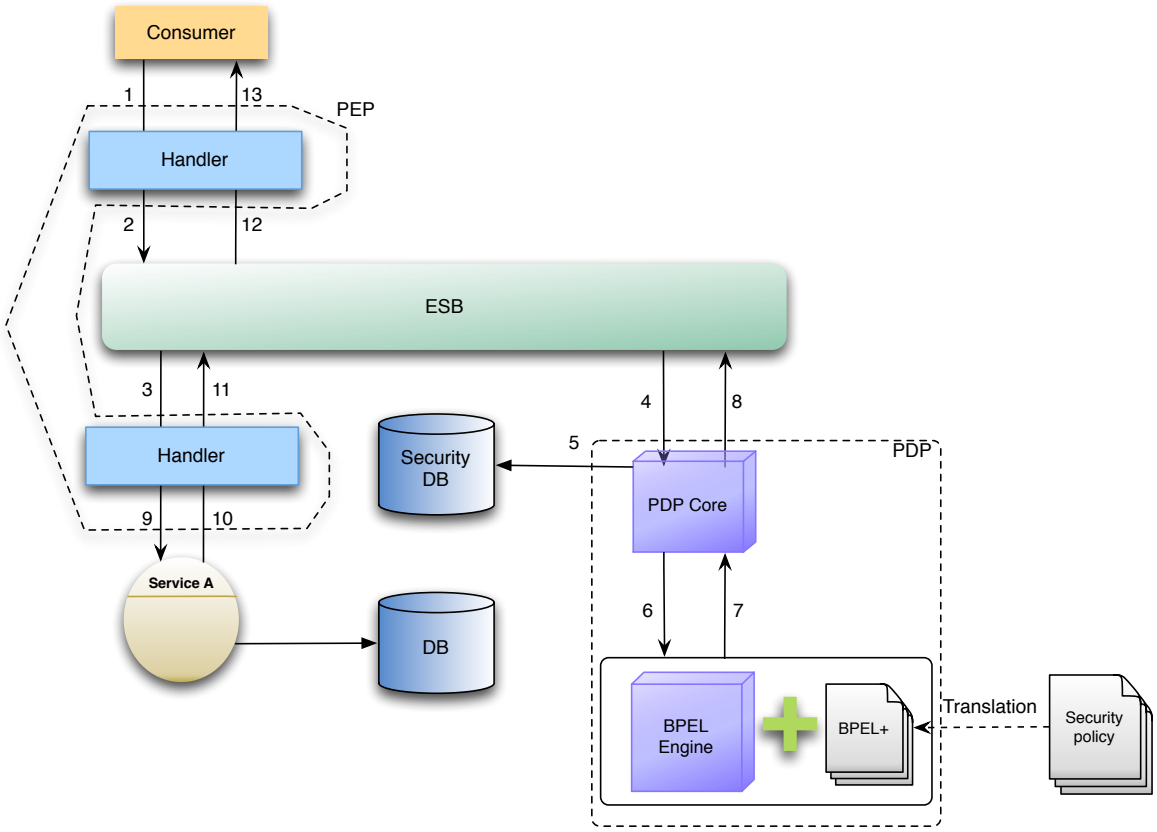

Figure 7: PEM deployed in the target architecture

be also done using standard protocols to pass credentials in a request. On the other side of the ESB, the handler attached to each Web service intercepts the request for the service. The security parameters embedded in the SOAP header are then extracted and a new request for autorisation is sent to the *security filter* which is essentially a BPEL process deployed as a Web service.

Figure 11 is the formal specification of the rule "clerks can perform all actions, except cancel and validate, for cheques not exceeding 10 000 $". This The policy is a choice of actions for the role `clerk` and for each action, the allowed values of the parameters are filtered. The `pick` activity of the BPEL language make it easy to implement the choice between actions.

When an autorisation's request for the Web service's operation `createCheque`, for example, the request is receive by the process in the step 1 of figure 12 using BPEL's `receive` activity. The process then computes a boolean which is the result of matching the actual values of the parameters and security parameters received from the consumer application to the specified values in the formal specification of the policy (step 2 of figure 12) using BPEL's `assign` activity. The last step is to respond to the request with a response containing the calculated boolean at the previous step (step 3) using BPEL's `reply` activity. This filtering process is what is required from the PDP at the third level (see section 1.2.3).

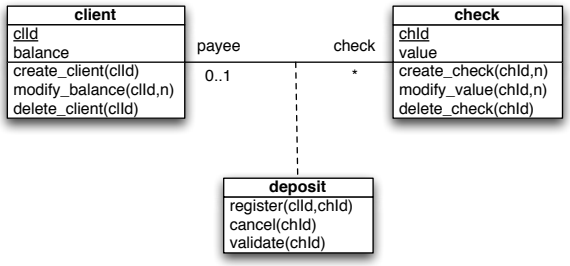

Figure 8: Class diagram of the bank IS [3]

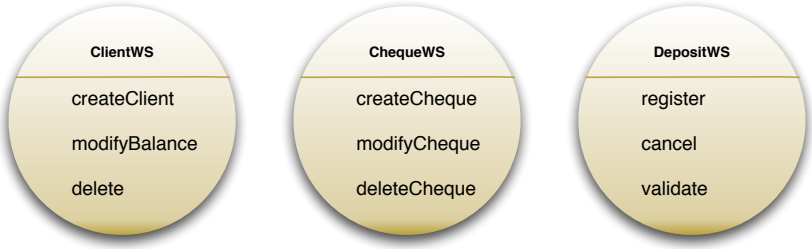

Figure 9: Web services in the bank IS

3.2 Securing an existing environment: the case of Med.e.com

3.2.1 Use cases

Figures 13 and 14 are sequence diagrams of the visualization of medical information and the request for medical expertise respectively.

SELKIS suggests a model driven security approach. In this scheme, when moving from an abstract model to a concrete model, we will need at some point a way to specify what are the characteristics of the network and the network equipments we are currently processing. In consequence, a network design tool may be a valuable asset that can be used to derive our abstract model to a concrete model. This tool, web-based, must be simple, easy to use with drag and drop facilities and export functions to some text format that can be processed by some external software.

A network discovery tool can also be required to facilitate integration.

The UML diagram in figure 15 shows how the authentication and the authorization servers integrate with Med.e.com application.

Both authentication and authorization servers are provided by Swid. These servers are compliant with OpenID, Security Assertion Markup Language 2 (SAML2) and XACML standards out-of-the-box. Some specific connectors to existing databases (for instance Lightweight Directory Access Protocol (LDAP)) must be implemented to benefit from existing users of the hospital and their associated rights.

At the Med.e.com software side, two solutions are envisioned. The first one consists in a software agent to be integrated in Med.e.com solution. The second one comprises a dedicated proxy

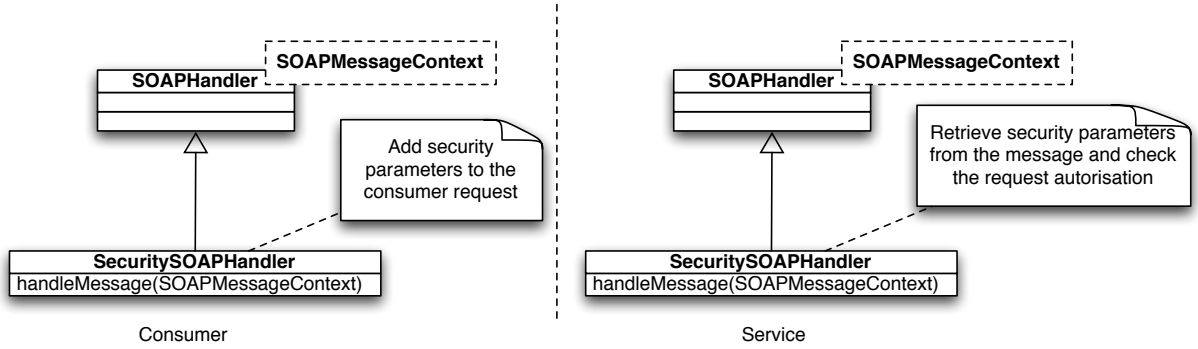

Figure 10: The PEP implemented as handlers

$$
\begin{array}{l}
\textbf{rule1}\,(\,) \triangleq \\
\quad (\,|\, ck \in CheckId, n \in 0..10000 \;:\; \langle \_, clerk, \_, \mathsf{create\_check}(\, ck, n\,)\rangle\,) \\
\quad\; | \\
\quad\quad |\, ck \in check : \\
\quad\quad\quad check(ck).value \leq 10000 \Longrightarrow \\
\quad\quad\quad\quad (\\
\quad\quad\quad\quad\quad (|\, n \in 0..10000 \;:\; \langle \_, clerk, \_, \mathsf{modify\_value}(\, ck, n\,)\rangle) \\
\quad\quad\quad\quad\quad | \\
\quad\quad\quad\quad\quad \langle \_, clerk, \_, \mathsf{delete\_check}(\, ck\,)\rangle \\
\quad\quad\quad\quad\quad | \\
\quad\quad\quad\quad\quad \langle \_, clerk, \_, \mathsf{register}(\, \_, ck\,)\rangle \\
\quad\quad\quad\quad)
\end{array}
$$

Figure 11: Formal specification of a rule using EB^{3}SEC [3]

in charge of authentication and authorization mechanisms, such that the Med.e.com software be modified only to retrieve user credentials from specific headers set by the dedicated proxy. This latter solution has the advantage of being less intrusive.

Med.e.com relies on some specific protocols like HL7 and DICOM. The PEM must be capable of embedding these specific messages in SAML2 messages if the implemented architecture requires so.

3.2.2 Orchestration

Orchestration of operations (or Web services) is an important topic in this project. The PEM must be able to orchestrate several processes. From an implementation point of view, the Policy Enforcement Manager must integrate a workflow component.

This workflow must offer a graphical editor and compliant with standard orchestration description languages (like BPEL).

3.2.3 Traceability

Watermarking technologies will be used to ensure traceability of data.

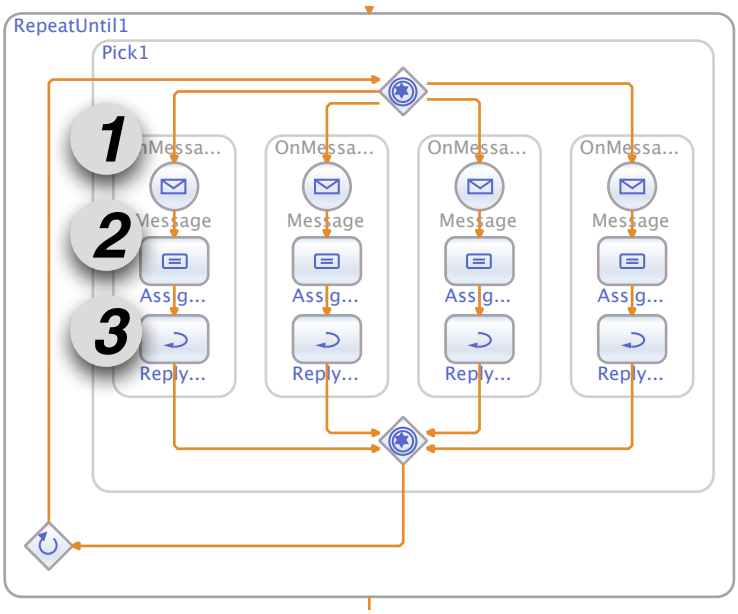

Figure 12: The policy `rule1` implemented as a BPEL process

4 Security properties and technical details

4.1 Integrity

To ensure integrity properties, all communications between components rely on Hypertext Transfer Protocol Secure (HTTPS). It is also possible to release some constraints by requiring that only communications with the user (and therefor, communications from an untrusted zone) be made over HTTPS.

Some signature based solutions could also be used for this project.

4.2 Authenticity and traceability

Authenticity and traceability are properties that are fulfilled using secure structures that are integrated into medical data. Data watermarking and dedicated readers will also be used when processing these kind of data.

4.2.1 Open-source

We must use as much as possible Open-source softwares to implement all these requirements. We must not be locked in by some vendors licenses if we want our implementations be successfully exploited by other teams.

4.3 Standards

As previously mentioned, relying on standards is a strong requirement in order to be independent from security software vendors.

In this project, we will use, for authentication protocols, OpenID due to its widespread acceptance and SAML2 considering its high level of security. XACML will be used for authorization

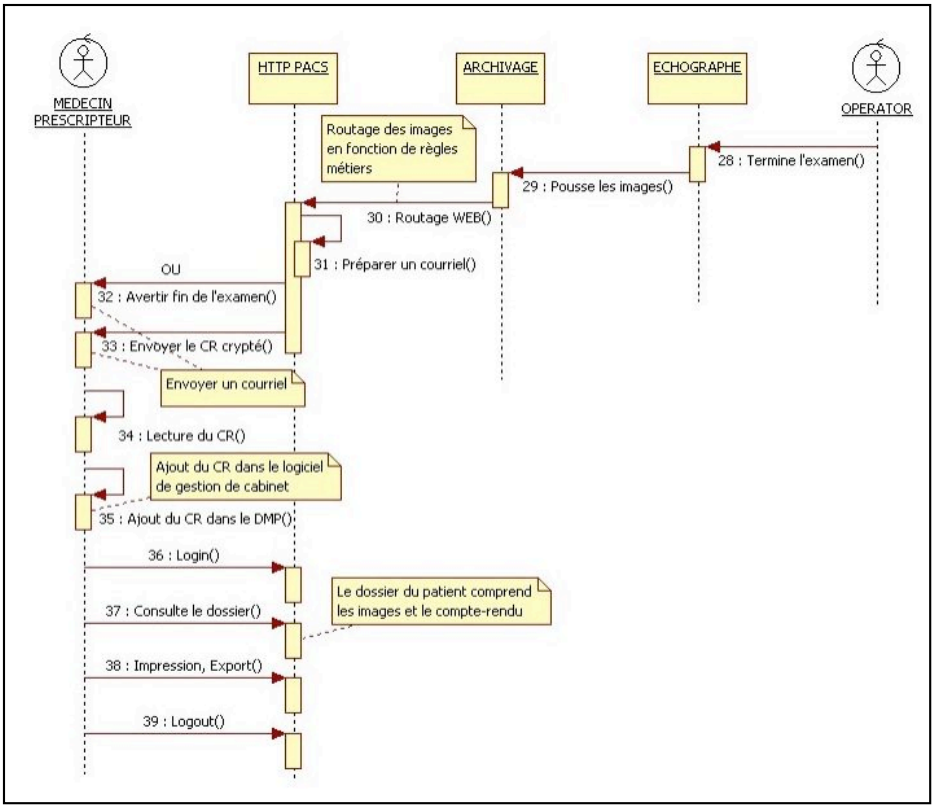

Figure 13: Medical information visualization

management. OpenID is a de-facto standard, while SAML2 and XACML are standards from the OASIS consortium. We may also investigate XACML 3, should it be ratified by OASIS during the project timeline.

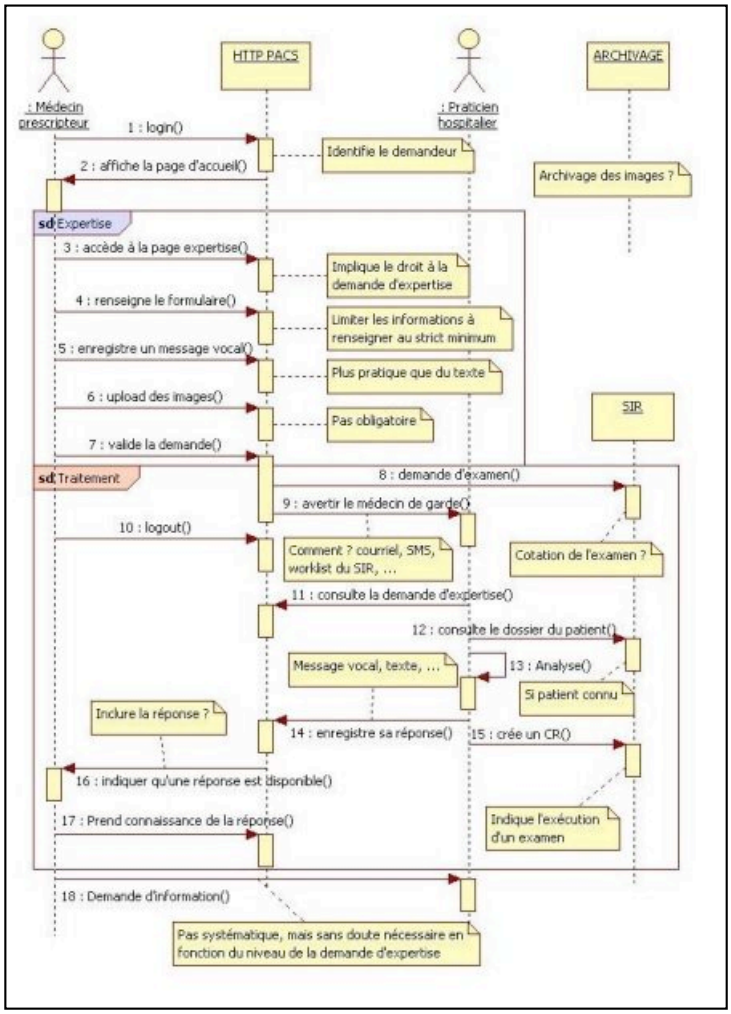

Figure 14: Medical expertise

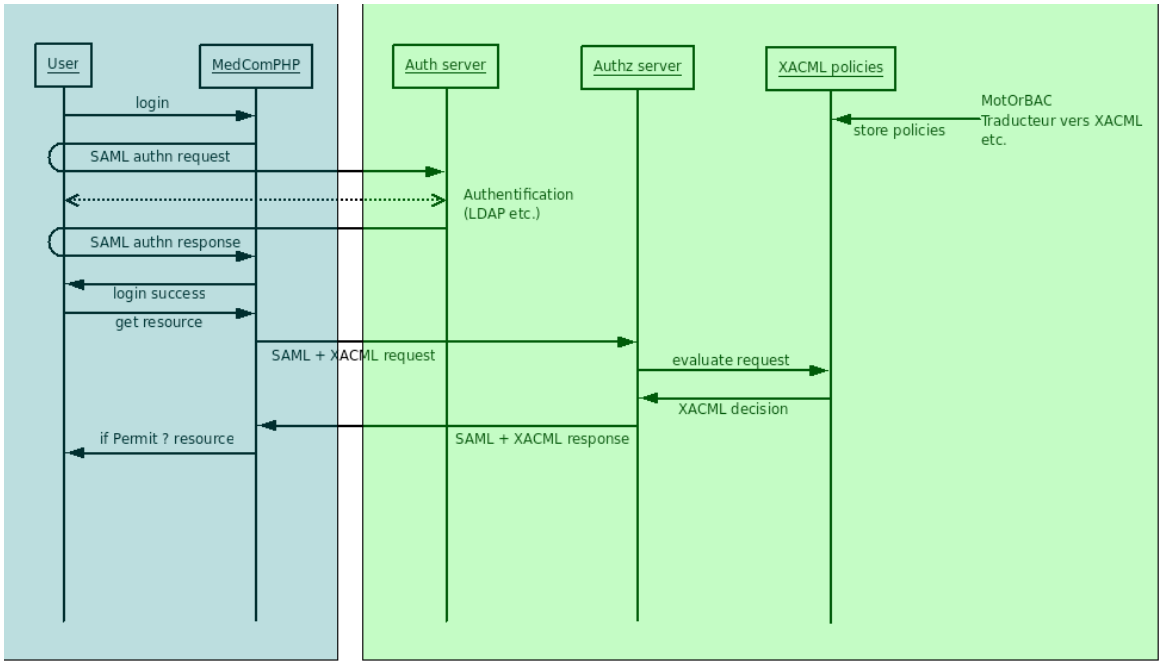

Figure 15: UML diagram

References

[1] A. Abou El Kalam, R. El Baida, P. Balbiani, S. Benferhat, F. Cuppens, Y. Deswarte, A. Miège, C. Saurel, and G. Trouessin. Organization Based Access Control. In *4th IEEE International Workshop on Policies for Distributed Systems and Networks (Policy'03)*, June 2003.

[2] M. Frappier and R. St.-Denis. EB^3 : an entity-based black-box specification method for information systems. *Software and System Modeling*, 2(2):134–149, 2003.

[3] Pierre Konopacki, Marc Frappier, and Régine Laleau. Modélisation de politiques de sécurité à l'aide dune algèbre de processus. In *INFORSID*, pages 295–310, 2009.

[4] OASIS. *eXtensible Access Control Markup Language (XACML) Version 2.0.* OASIS, 2005.

[5] OASIS. *Web Services Business Process Execution Language Version 2.0.* OASIS, 2007.