On federated single sign-on in e-government interoperability frameworks

Giansalvatore Mecca*, Michele Santomauro, Donatello Santoro and Enzo Veltri

Dipartimento di Matematica, Informatica ed Economia,
Università della Basilicata – viale dell'Ateneo Lucano,
10 – 85100 Potenza, Italy
Email: giansalvatore.mecca@gmail.com
Email: michele.santomauro@gmail.com
Email: donatello.santoro@gmail.com
Email: enzo.veltri@gmail.com
*Corresponding author

Abstract: We consider the problem of handling digital identities within service-oriented architecture (SOA) architectures. We explore federated, single sign-on (SSO) solutions based on identity managers and service providers. After an overview of the different standards and protocols, we introduce a middleware-based architecture to simplify the integration of legacy systems within such platforms. Our solution is based on a middleware module that decouples the legacy system from the identity-management modules. We consider both standard point-to-point service architectures, and complex government interoperability frameworks, and report experiments to show that our solution provides clear advantages both in terms of effectiveness and performance.

Keywords: web services; SOA; service-oriented architecture; SSO; single sign-on; federated identity; government interoperability frameworks.

Reference to this paper should be made as follows: Mecca, G., Santomauro, M., Santoro, D. and Veltri, E. (2016) 'On federated single sign-on in e-government interoperability frameworks', *Int. J. Electronic Governance*, Vol. 8, No. 1, pp.6–21.

Biographical notes: Giansalvatore Mecca is a Full Professor of Computer Science and Computer Engineering at the University of Basilicata. He received his PhD from the Sapienza University in Rome. He has been with the University of Basilicata since 1995. His research interests include data-cleaning, data-integration, information extraction, web-service interoperability and SOA.

Michele Santomauro is a Research Associate in Computer Science at the University of Basilicata. His research interests include web service interoperability, SOA, and government interoperability frameworks.

Donatello Santoro is a Research Associate in Computer Science at the University of Basilicata. He got his PhD from the University Roma Tre. His research interests include data-cleaning, data integration, and web service interoperability.

Enzo Veltri is a PhD student in Computer Science at the University of Basilicata. His research interests include data cleaning, web service interoperability, SOA, and government interoperability frameworks.

1 Introduction

The development of an electronic public administration (PA) is a priority for the majority of governments in the world. The ultimate goal is to promote a new paradigm of society, the *information society*, and to make the PA more efficient, effective, and citizen-centred (Soares and Amaral, 2011). This goal imposes a fundamental and non-trivial technical requirement to PAs, namely the ability to work in a cooperative way. This, in turn, requires the existence of interoperability frameworks for their information systems (IS). Interoperability can be defined as the ability to exchange data and services by different computing systems. A possible way to achieve interoperability is by adopting a *service-oriented architecture* (SOA) (Krafzig et al., 2005), where a *service consumer* requires data or applications (also called services) offered by a *service provider*, with an exchange of messages through web services organised according to a common format.

Typically, SOA solutions are based on *point-to-point architectures*, in which each consumer is responsible of discovering the relevant services, and to communicate with their providers in a direct way. This paradigm shows a number of limitations in complex organisations and in e-government frameworks that need to coordinate multiple agencies. In these scenarios, several different technologies are usually deployed, so that standardising the way services are exchanged becomes a critical success factor. These problems are usually solved adopting *government interoperability frameworks* (GIFs). Some of these framework rely on *middleware-oriented solutions*, where systems do not interact directly with each other, but rather with intermediate layers of software. These middleware components are responsible for various services, among which message-format standardisation, transparent routing, enhanced security, reliability, and quality-of-service (QoS) monitoring.

Since the cooperation often involve the exchange of sensible information, service providers must protect their own resources against unauthorised access. If from one side the problem of protecting the communication between systems is typically solved using well-established standards (as TLS and SSL), on the other side the problem of regulating the access to distributed resources is more complex, since it requires suitable means for identification, authentication and authorisation of a specific user in a system (Bertino et al., 2009).

Identification is the process of acquiring the user credentials, e.g., username, certificate, smart card. *Authentication* is the process of verifying that the user really is who he or she claims to be, by checking their credentials, e.g., password, against a database of recognised identities. Finally, once a user is identified and authenticated, *authorisation* is the process of deciding whether the use is allowed to access a resource he or she requests.

Deploying security within interoperability solutions requires to solve several technical problems. First, the security infrastructure is distributed and composed of different modules. The overall security level of the system is the same as that of its weakest part. As a consequence, ensuring security in a distributed environment often requires to modify existing systems, and this comes with a cost. Second, systems are heterogeneous, and a common security standard needs to be identified. Finally, a natural requirement in these

distributed applications is that information systems share user authentications, in order to avoid that a user needs to provide his or her credentials multiple times. A domain of systems that need to share user authentications is called a *federation of system*.

To summarise, we can identify three key requirements in a security architecture:

i it should have limited impact on existing systems

ii it has to preserve interoperability across different environments

iii finally, it should allow systems to share user authentications.

A possible solution to these problems is to adopt a *single sign-on* (SSO) architecture. The main idea is to move all the security logic in a dedicated component, called *SSO service*, and thus relieve other parts of the system from security obligations. In SSO architectures all security algorithms are centralised in the SSO server, which acts as the single authentication point for a whole domain. A user is required to sign-on only once, and then his or her credentials and attributes are stored for the length of the session in such a way that he does not need to provide them for subsequent accesses, even though he or she may be interacting with many different secure elements within a given domain.

To give an intuition, we can describe a simple SSO scenario as follows. The user, using a client (such as a browser), first authenticates her/himself in the particular domain using the SSO server. To do this, he or she provides the correct authentication credentials, e.g., username/password. The SSO server validates the users' credentials using the underlying security infrastructure, and releases a *token* to the requesting user. From then on, the token can be used to request services to other applications (these steps will be illustrated in more detail in the next section).

The advantages of a SSO architecture are evident:

- implementation, deployment and maintenance of existing systems are simplified, since the communicating parties in the distributed system do not need to individually implement all of the security features.
- the adoption of web service standards into the SSO server (typically, a web service itself) makes the SSO architecture interoperable with different underlying technologies.
- the SSO authentication model enhances the overall security, since the only point that accepts security credentials is the SSO server itself, and therefore credentials are not unnecessarily transmitted between systems.

In addition to these, another prominent benefit of SSO architectures is that it enables independent organisations to extend the capabilities of their existing identity-management services, creating a *federated single sign-on (FSSO)* solution. FSSO is the prominent solution to guarantee a secure interaction between different domains, each with its own security component. The SSO servers are integrated within a so called 'circle-of-trust', a sort of a formal contract that guarantees that all activities in a specific domain are valid also by authorities belonging to trusting domains.

This paper studies the problem of integrating information systems within SOA architectures with FSSO identity management, a problem that is significantly more challenging than deploying standard SSO solutions (Sliman et al., 2009). We consider both

point-to-point service architectures, and complex middleware-oriented frameworks. The main contributions of the paper are the following:

1 we develop a federated solution to protect services, both on the service requester and service provider level, that preserves the requirement of autonomy of the existing domains

2 we introduce an identity management module, called FreESBee-SP, that acts a middleware layer between the local information system and the SSO components used to authenticate users and authorise message exchanges

3 we report a comparison between a middleware-based solution and a standalone solution, and a discussion on benefits and disadvantages of both approaches.

The paper is organised as follows. We first review some related work in Section 2. Then, we report an overview of the prominent standards for SSO, and concentrate on the main components of the SAML standard (Section 3). Then, we analyse how to deploy the SAML architecture within a distributed environment (Section 3), and introduce FreESBee-SP, a middleware module to ease the integration of information-systems within SOA architectures with SSO identity management (Section 4). Finally we show the results of a comparison experiment between a middleware-based solution and a standalone one (Section 5).

2 Related work

The wide adoption of distributed architectures has emphasised the importance of security in message-oriented applications. Despite the availability of well established standards, the problem of regulating access to distributed resources is still open.

In the last years, several solutions for the SSO have been proposed. Some examples are the central authentication service (CAS) protocols (Aubry et al., 2004), OAuth/OAuth2 (Sun and Beznosov, 2012) and OpenID (Recordon and Reed, 2006). The consolidated standard in this field is the OASIS security assertion markup language (SAML), that define a general purpose solution to the digital identification and authorisation mechanism, that can be integrated in every environment.

The SAML standard is an ideal solution for SSO within a SOA environments. However there are no standard ways or guidelines that describe how to protect a service. This leaves ample freedom in the design of solutions, as described in Sliman et al. (2009), Ates et al. (2007) and Emig et al. (2007).

Sliman et al. (2009) describe a solution for the dynamic composition of services with the support of the PEtALS[1] enterprise service bus (Schmidt et al., 2005). In order to produce a standard-based solution and guarantee the interoperability with other systems, authors embed SAML components inside the PEtALS architecture. In this way the system is able to dynamically create and protect resources on the fly. The benefit of this solution is that the system is able to protect not only resources, but also the middleware ESB infrastructure. Similar to our approach, the security layer is completely transparent to services. The main drawback is that this layer is strictly coupled with the PEtALS implementation, limiting its adoption inside other implementations like Apache ServiceMix[2] or Mule ESB.[3] The approach presented in our paper is a separated component, that can be used with all kind of ESBs.

Ates et al. (2007) deal with the interoperability between two different domains that adopt different standards for protecting resources. Authors describe the real case of several domains (called federations) that need to exchange data, each of them with its own mechanisms to regulate access to resources. The work is focused over the SAML and the WS-Federation standards and describes a solution based on the adoption of third party component that act as a proxy between both standards. Our solution, instead, is based on the assumption that all domains adopt the SAML standard architecture.

Another approach (Emig et al., 2007) defines a custom solution focused around the concept of 'identity as a service'. Here, the authentication and authorisation process is managed entirely by means of web services: the security token service (STS) component and the policy decision point (PDP) component, that respectively perform a check about the identity of the requester and the permissions that it has over the resources. All interactions with services are regulated by these two components. The main benefit is that the architecture is drastically simplified, but there are several drawbacks. First of all, the architecture can be deployed only within a completely distributed environment; moreover, it is necessary to modify existing services in order to provide a check about the requester; and finally this is an ad-hoc solution that does not rely completely on standards technologies, with the consequence that it is not interoperable with standard-based systems.

3 SSO and SOA: an embedded approach

In this section, we introduce a first SSO architecture that we call the *embedded architecture*. The core idea is to implement the security logic into the service consumer and service provider. In order to do this, we rely on two well established standards, namely *security assertion markup language (SAML)* and *extensible access control markup language (XACML)* that define the identification and authorisation protocol, respectively. These are introduced in the next sections.

3.1 The SAML standard

The *security assertion markup language (SAML)* (Ragouzis et al., 2008), developed by the OASIS consortium, is based on open standards promoted by the W3C, like SOAP, XML, XMLSignature, XMLEncryption, and WS-Security.

SAML is designed to provide a flexible way to specify *user attributes*. In fact, modern applications usually need additional information about users to make proper authorisation decisions. These user attributes are especially useful in multi-organisation scenarios where an application does not have access to full user profiles through other means, such as directory services.

The SAML standard defines the SSO process in terms of message exchanges between three main actors: the *Principal*, the *Authorities* and the *Service Provider*.

- The *Principal* is the user that starts the business process.
- The *Authorities* are responsible of releasing data about the identity and the profile of the Principal. In many cases this information are provided by several components: the *identity provider (IdP)* and one o more *attribute authorities* (AA). Intuitively the IdP maintain user identities, while the roles that a user may play are stored in the Attribute Authorities. Usually each user can have more than one profile, e.g., citizen profile, professional profile, school profile etc.

- The *service provider (SP)* decides to grant or deny access to the resource to a particular Principal. In order to do that, it act as a mediator between principals and authorities.

The interaction between these parties requires several exchanges of XML messages, that are described in Figure 1, and work as follows.

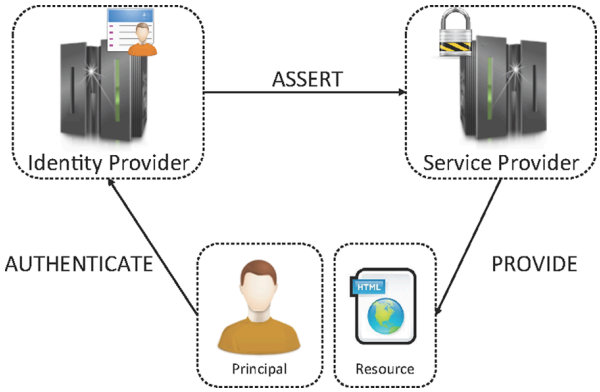
i The principal requests a resource from the service provider. Since it needs to know the identity of the requester, the SP produces an *authentication request* for the IdP and redirects the principal to it.

ii If the user is successfully identified, the IdP returns to the SP a certified statement about the identity of the Principal, under the form of an *authentication response*.

ii For authorisation purposes, the SP needs to acquire a profile for the user. It sends an *Attribute Request* to the AA. The AA returns a profile for the Principal under the form of an *attribute response*.

The combination of authentication and attribute data are usually called *SAML assertions*. Using SAML assertions the service provider is able to evaluate the principal's attributes with the access control policies and it grants or denies access to the requested resource.

Figure 1 SAML interaction (see online version for colours)



As a result of this process, an *authorisation assertion* statement is released to the principal. From now on, all other requests that will be submitted from the same principal to the service provider, even for other resources, can be directly evaluated without the need of repeat the identification process, since the identity of the requester is already known to the service provider.

Since the publication of the standard, several implementations, both closed and open source, have been released. In this context, we can identify as a *de-facto* standard the *Shibboleth system* (Morgan et al., 2004). This is an open source SAML compliant project started in 2000 and developed by the Internet2 consortium with the goal of providing a ready-to-use SSO solution for all resources belonging to a specific domain. The implementation includes two major software components: the Shibboleth identity provider and the Shibboleth service provider. Since Shibboleth is oriented to web-based client-server architecture, it is relatively simple to protect resources as webpages, thanks to its integration

with the popular Apache2 web server. However, adopting Shibboleth in a SOA environment requires some particular treatment, that we will discuss in the next sections.

3.2 The XACML standard

The *extensible access control markup language* defines a declarative access control policy language based on XML and a processing model describing how to evaluate access requests according to the rules defined in policies (Aa.Vv, 2013). It is another protocol defined by the OASIS consortium and it is often used in combination with SAML. The two solutions are independent from each other and can be implemented separately, but together they are able to guarantee a complete authentication and authorisation mechanism.

The authorisation process involves four main components:

- a *policy administration point (PAP)* that manages policies for the access authorisations
- a *policy decision point (PDP)* that takes the decision of granting or denying access to the requested resource
- a *policy enforcement point (PEP)* that intercepts the access request to a resource, submit it to the PDP and grants or denies access depending on the decision of the PDP
- a *Policy Information Point (PIP)* that acts as a source of attribute values.

The execution flow, described in Figure 2, starts with the PEP that receives an access request and forward it to the PDP. The PDP submits a request to the PAP in order to retrieve the policy defined for the access to the resource, and then it sends a request to the PIP to retrieve the profile of the requester. Inside the XACML context a profile is organised in three main sections:

a subject, the entity that request attributes

b resource, the element to which the subject wants to access

c environment, section for additional informations.

When the policy and the profile are collected by the PDP, it proceeds with the evaluation and decides to grant or deny access to the resource. This decision is then forwarded to the PEP that will redirect the original request to the resource. The result of this process is an *authorisation decision statement*, asserting that a subject is permitted to perform a specified action on a given resource.

Similar to SAML, this standard is XML-oriented. This facilitates its integration within SOA solutions.

3.3 The embedded architecture

SAML and XACML provide the technical tools to implement our embedded sign-on solution (Sliman et al., 2009; Emig et al., 2007), as shown in Figure 3.

On one side we have the service consumer, i.e., the system that interacts with the user, and on the other side there is the system that provides a service (service provider). The communication between these systems, as illustrated in Section 1, is performed using an exchange of standard XML messages.

Figure 2 XACML component interaction (see online version for colours)

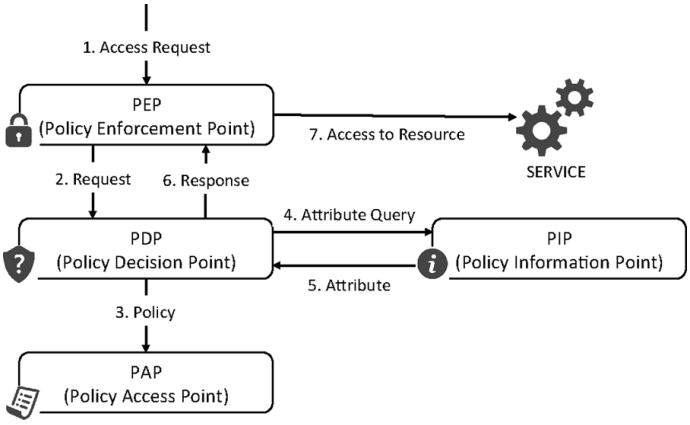
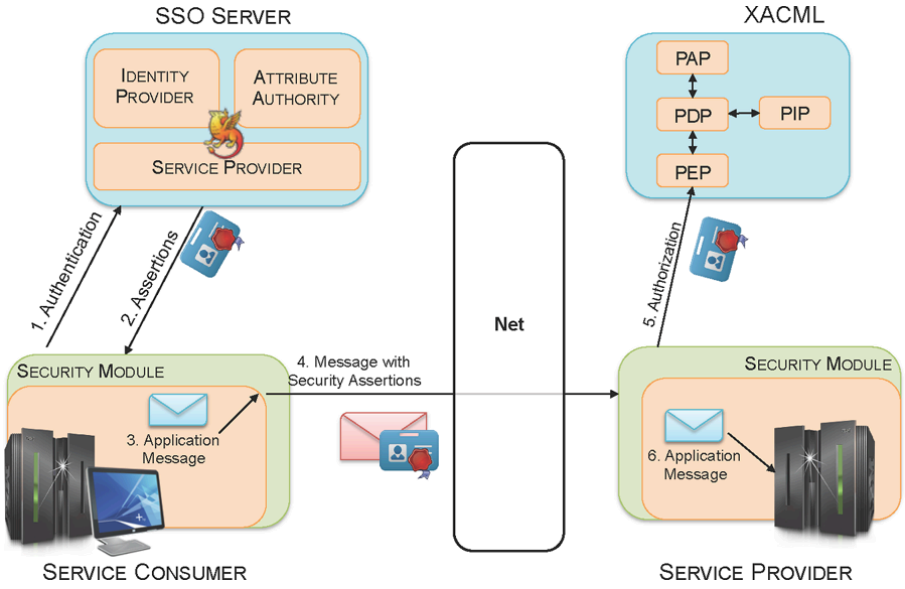


Figure 3 An embedded approach to single sign-on in a SOA architecture (see online version for colours)



Before starting the request, (1) the service consumer begins the user authentication process interacting with the SSO Server, in order to acquire the SAML assertions of the user (2), as described in Section 3.1. Moreover, these assertions are typically signed by the SSO Server in order to prevent further modifications.

Once the authentication process is completed, the user interacts with the system and generates an application request message (3). Since both application message and SAML assertions are in XML format, they are easy to combine generating a message with security assertions (4), that is sent to the service provider, that can be located in a new domain.

To guarantee that the overall transaction is secure, not only the service consumer, but also the service provider need to perform security checks. Usually different interoperability

domains use independent SSO servers. As a consequence, a main security requirement is that user assertions are maintained across domains, i.e., authentication is performed locally, but assertions can be used to check identity even in a different domain (Jensen, 2011). This enhance the security of the service provider, that can perform an authorisation control (5), for example using the XACML standard process, before executing the service (6).

It can be seen that this solution effectively guarantees the complete autonomy of services. However, it has a main drawback: it imposes to change the application logic of local information systems. More specifically, developers typically need to recode the login code of the systems in order to orchestrate the interaction with the SSO modules. In addition, SOAP messages need to be properly enriched with assertions. Assertions, in turn, must be processed both on the consumer and provider side. It is easy to see that in an environment with hundreds of existing services, this can become extremely expensive and time consuming.

4 The FreESBee-SP solution

To alleviate the problems of the embedded solution, in this section we introduce an alternative, middleware-based solution, called FreESBee-SP. Our component is open source and publicly available.

FreESBee-SP is a SAML 2.0 and XACML compliant solution belonging to the Middleware project, a complete suite of tools that enable interoperability between the Italian public administrations (Baldoni et al., 2008). It is designed to add security features to existing systems, without the need of radically changing their internal architecture. It can be used both in point-to-point scenarios, and middleware-based GIFs environments.

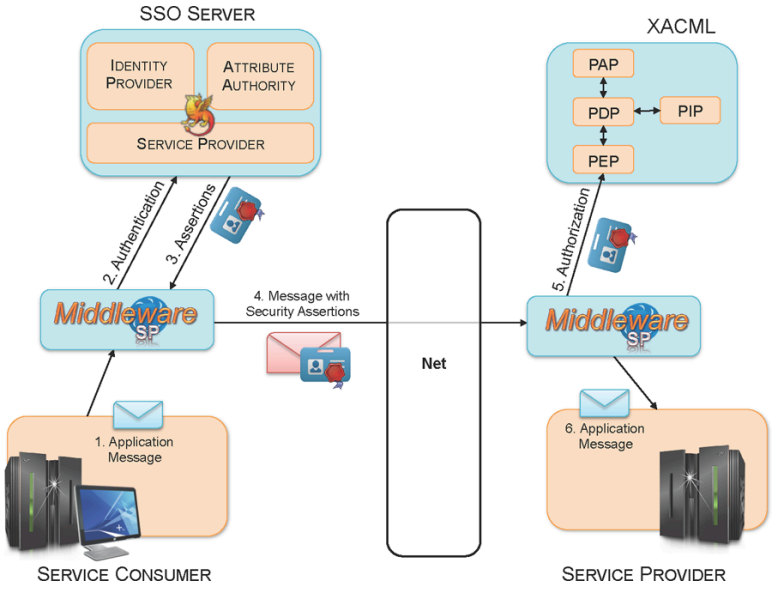
Figure 4 illustrates the process of protecting a service using FreESBee-SP. On the requester side, it works in strict cooperation with the Shibboleth SAML implementation that is used to identify the requester, collects the list of attributes belonging to its profile and checks for the authorisation. On the provider side it rechecks the authorisation over the incoming profile, in order to be sure that no changes happen during the transmission of the message. Because of differences between the requester and the provider side protection, we decided to split the description of details in two different paragraphs.

4.1 Requester-side protection

The protection of a requester information system is performed in conjunction with a Shibboleth instance. In this case, FreESBee-SP acts purely as a middleware component and leaves task of checking the requester identity and authorising its transaction to Shibboleth's IdP and AA. FreESBee-SP orchestrates all exchanges between the IdP, AA, and the service provider in order to acquire the statements that certify the identity and the profile of the requester.

The user starts a transaction by using the local system to generate an application request. The main difference wrt to the embedded approach is that the system does not need to interact with the SSO server anymore. In order to start the authentication process, the Service Consumer sends the message to FreESBee-SP instead. Notice that this redirection merely requires to change an endpoint, usually through a configuration file, and has no major impact on the system's codebase. As a consequence, it significantly lowers the cost of deploying the solution.

Figure 4 The FreESBee-SP solution (see online version for colours)



When FreESBee-SP receives the message, it checks on the local cache if the user is already authenticated, otherwise it redirects the request to the Shibboleth components. At the end of the authentication process, the SAML assertions are returned to FreESBee-SP that stores them in its local cache. For the next requests of the same user these cached assertions will be used, without contacting the SSO server.

As a final step, the request, properly enriched with SAML assertions, is sent to the service provider.

4.2 Provider-side protection

In order to avoid that malicious attackers directly access the resource by skipping the security checks on the requester side, it is also necessary to verify the user authorisations on the provider side. FreESBee-SP provides support for this operation as well. The strategy in this case is similar to the NAT technique used inside networking: the idea is to hide the real endpoint of the resource behind a *protected endpoint* published by FreESBee-SP.

For every message received on this endpoint, the system extracts the SAML assertions inside the message and the authentication process starts. As first step, FreESBee-SP checks if the assertions are properly signed by a *trusted* service provider. If so, the XACML module verifies if the user is allowed to contact the resources wrt local ACL policies, as described in Section 3.2. At the end of the evaluation process, the PDP response is returned to FreESBee-SP. It can reject the request, by returning an error message with an explanation, or can accept it, and forward it to the service provider.

Also in this case, the deployment does not require changes to the existing code.

5 Experiments

In this section, we report experimental results in order to empirically evaluate the impact of FreESBee-SP in terms of performance and resource consumption in SSO scenarios. Our baseline is the embedded solution.

The use-case we choose is a real-life web service about transportation: an employee of the *Driver and Vehicle Licensing Agency* wants to get information about a car owner using a web application. These data are stored in a separated repository, that can be contacted through a web service endpoint. Using the terminology introduced in the previous sections, the employee is our *Principal*, the client web application is the *Service Consumer*, and the web service represents the *Service Provider*.

We develop two implementations of this scenario. In the first, *embedded implementation*, we handle digital identities by adding the appropriate logic to the web application. In the second, *middleware-based implementation*, we leave the web application untouched and we add the FreESBee-SP component between the service consumer and provider.

5.1 Test environment

The goal of these tests is to verify the impact of our middleware component on the architecture, in terms of scalability and resource consumption. We measure scalability by number of message transactions per seconds that are completed. To simulate the traffic originated by an increasing number of clients, we used Apache JMeter v. 2.12, configured to simulate a variable number of concurrent clients, ranging from 1 to 100. Each of these clients simulates an employee that starts a request from the service consumer, for a total of 100.000 messages sent for each test session. This experiment involve a total of five machines, two physical machines (PM1 and PM2) and three virtual machines (VM1, VM2 and VM3) organised as follow:

- JMeter is on the first physical machine PM1
- the web application and the web service are hosted within the same application server, Apache Tomcat 6.0.36, inside the dedicated physical machine PM2
- the identity provider and the attribute authority are from the Shibboleth 2.3.8 implementation, installed within the application server Apache Tomcat 6.0.36 inside the virtual machine VM2
- the Shibboleth Service Provider 2.5.0 is inside the virtual machine VM1
- FreESBee-SP 1.0 is inside the virtual machine VM3, together with the LDAP repository (based on OpenLDAP 2.4.28) and the PostgreSQL 8.4 DBMS.

Physical machines are quad-core Intel i3 Processor at 2.5 GHz with 4 GB of ram and Windows 7 64 bit as operating system, while virtual machines were hosted on dual-core machines with an Intel Xeon Processor at 3GHz. Virtual machines were assigned 100% of CPU usage and 2.5 GB of RAM. All machines were connected to a dedicated high-speed Gigabit Ethernet switch, in order to limit the latency and avoid slowdowns caused by the network. The operating system is Ubuntu Server 12.04 LTS; the recorded values were reported by the Summary Report of JMeter and resource consumption was measured with the built-in profiler provided with the distribution. We modified the Tomcat configuration,

increasing the maximum number of threads to 1000 and also changing the base memory to 1 GB.

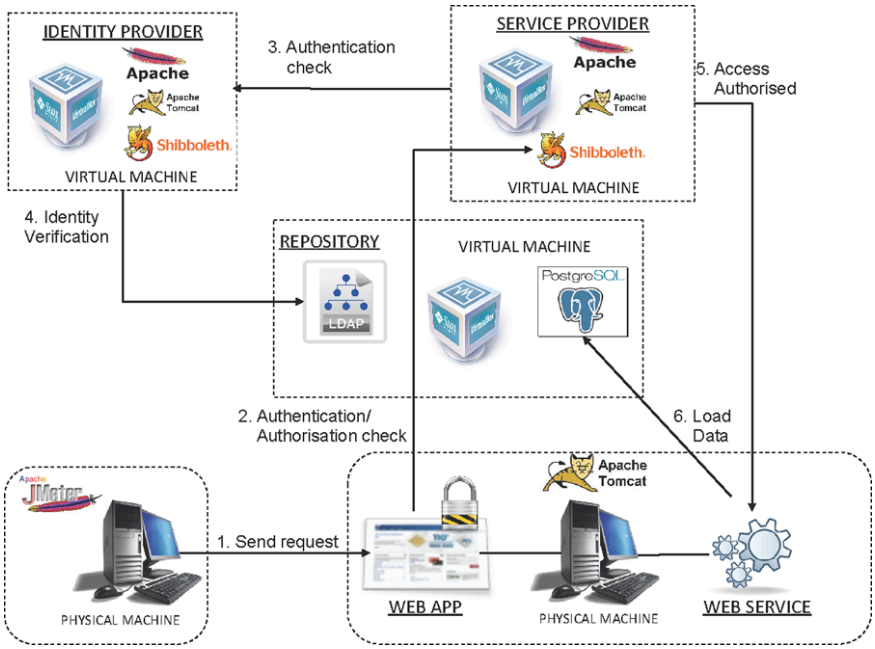
Using these machines, we simulate three different configurations:

i an Embedded implementation, shown in Figure 5, that use the approach described in Section 3

ii a Middleware implementation, detailed in Figure 6, that relies on FreESBee-SP, as discussed in Section 4

ii a No-Security implementation, were no security or identity management features were implemented.

Figure 5 Deployment of the embedded scenario (see online version for colours)



5.2 Test results

We first test scalability. Results are shown in Figure 7. As it can be seen in Figure 7(a) chart, the Embedded implementation has better performance wrt the Middleware one. This is somehow expected, since the introduction of FreESBee-SP makes the web application easier to develop and integrate, but requires more HTTP requests, that negatively impact the overall scalability. The main cause of this is the additional latency due to authentication and authorisation requests. As a consequence, the embedded solution shows a throughput that is steadily higher than the one of the middleware solution, with a peak difference between 10 and 20 concurrent clients. An important element that should be highlighted is that the entire architecture scales nicely up to 100 concurrent clients.

Figure 6 Deployment of the middleware scenario (see online version for colours)

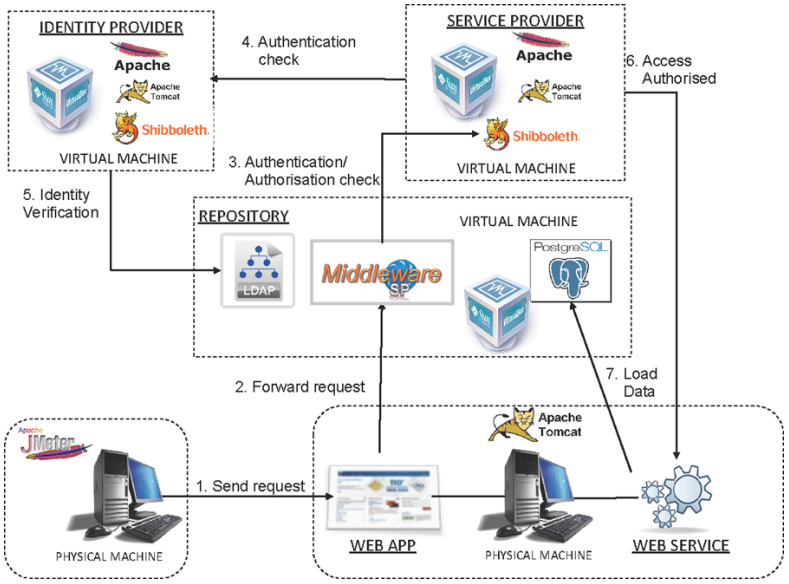
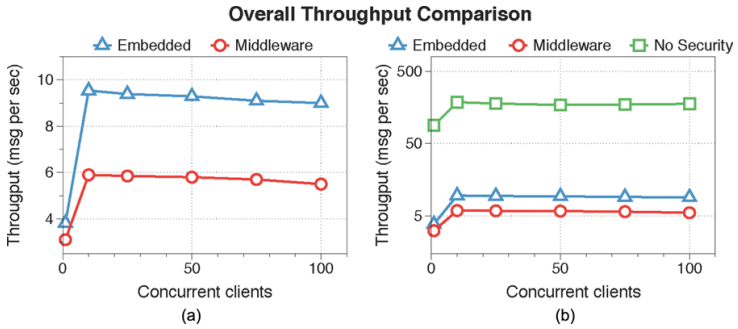


Figure 7 Throughput variation with increasing concurrent clients: (a) embedded vs. middleware solution and (b) with and without security constraints (see online version for colours)



Another element is that the FreESBee-SP implementation gives more flexibility in terms of deployment. For example, in order to avoid that the middleware component becomes the bottleneck of the architecture, it is possible to balance its load over several machines.

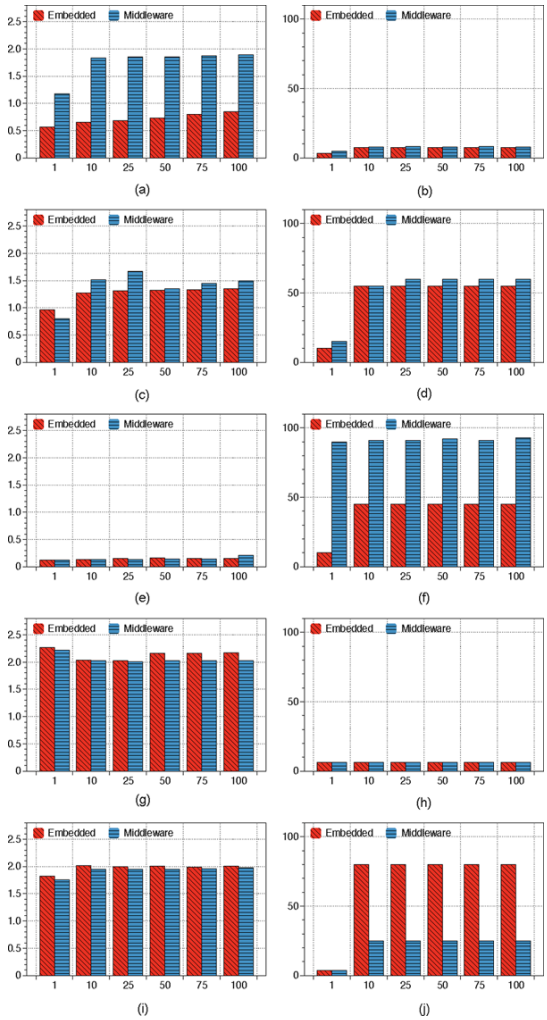
Figure 7(b) shows a comparison wrt the No-Security implementation. Notice that, in the no-security configuration, transactions are over 10x times faster than security-enabled solutions (notice the logarithmic scale). We were able to measure an average of 180 processed messages per second (with peaks of 400). Notice, however, that the one we report are worst-case results, since our experiment is designed in such a way that at every request the user authenticates himself. This is highly unlikely in practise, due to the SSO feature of the architecture.

Combined, Figure 7(a) and (b) show that, while it is true that the middleware solution suffers from a degrade in scalability wrt the embedded one, this decrease is less significant

when compared to the inherent reduced scalability of security-enabled solutions wrt no-security ones.

Finally, for every execution we also measured resource consumption on each machine, in order to identify possible bottlenecks in our test architecture. Figure 8 provides an aggregate vision of the CPU usage and RAM usage for all machines. Also in this case, results confirm that the components that require more resources are the ones related to security. In particular, there are significant differences between the PM2 and the VM3 machines, that run the web application with the security module, and the FreESBee-SP component, respectively. In fact, in the Embedded implementation the Apache Tomcat application server needs an higher amount of memory and CPU time in order to handle the security features.

Figure 8 Resources usage with increasing concurrent clients: (a) VM1 – RAM usage (in GB); (b) VM1 – CPU usage (in %); (c) VM2 – RAM usage (in GB); (d) VM2 – CPU usage (in %); (e) VM3 – RAM usage (in GB); (f) VM3 – CPU usage (in %); (g) VM4 – RAM usage (in GB); (h) VM4 – CPU usage (in %); (i) VM5 – RAM usage (in GB) and (j) VM5 – CPU usage (in %) (see online version for colours)



Overall, we notice that in all conditions, also with 100 concurrent clients, nor RAM nor CPU resources were saturated.

To summarise, we believe our result confirm the effectiveness of a middleware-oriented solution such as FreESBee-SP. In fact:

i on the one side, it is true that the more complex nature of the transactions due to the middleware modules reduces scalability, but this reduction is moderate when compared to the one due to handling security in the first place.

ii at the same time, the middleware-based solution significantly alleviates the burden of integrating legacy information systems within the FSSO solution, since it does not require extensive coding, nor software maintenance.

6 Conclusions

This paper discusses the issue of federated single-sign-on solutions within e-government frameworks. We provide a general introduction to the problem, and then compare two solutions to the problem of adding authentication and authorisation capabilities to federated SSO environments.

The first, naive solution consists of implementing FSSO standards directly into the service consumer and service provider. The second solution is conceived to reduce the effort needed to integrate legacy systems. It is based on a middleware component, called FreESBee-SP, that can be used both in point-to-point and middleware-oriented GIF environments.

We report an empirical evaluation of the impact of FreESBee-SP within distributed architectures. By comparing scalability and resource consumption of the two solutions, we show that, thanks to the better organisation and distribution of components, the adoption of a middleware module for FSSO can bring significant benefits in terms of development cost, while suffering a moderate reduction in performance levels.

As future work, we plan to enrich our suite of middleware modules to facilitate extensive tests of complex FSSO solutions, with multiple clients and multiple federated domains. In these scenarios, it is important that the federated solution be tested even in the early stage of its development, in order to identify architectural and security pitfalls. Dedicated modules are needed for this purpose.

References

Aa.Vv (2013) *OASIS eXtensible Access Control Markup Language (XACML) TC.*

Ates, M., Gravier, C., Lardon, J., Fayolle, J. and Sauviac, B. (2007) 'Interoperability between heterogeneous federation architectures: illustration with saml and ws-federation', *Third International IEEE Conference on Signal-Image Technologies and Internet-Based System, SITIS 2007*, IEEE Computer Society, 16–18 December, Shanghai, China, pp.1063–1070.

Aubry, P., Mathieu, V. and Marchal, J. (2004) 'Esup-portail: open source single sign-on with cas (central authentication service)', *Proc. of EUNIS04-IT Innovation in a Changing World*, 19 June–2 July, Ljubljana, Slovenia, pp.172–178.

Baldoni, R., Fuligni, S., Mecella, M. and Tortorelli, F. (2008) 'The Italian e-government enterprise architecture: a comprehensive introduction with focus on the sla issue', *5th International Service Availability Symposium on Service Availability, ISAS 2008*, Springer, Tokyo, Japan, pp.1–12.

Bertino, E., Martino, L., Paci, F. and Squicciarini, A. (2009) *Security for Web Services and Service-Oriented Architectures*, Springer Science and Business Media, Springer Heidelberg Dordrecht, London, New York.

Emig, C., Brandt, F., Kreuzer, S. and Abeck, S. (2007) 'Identity as a service–towards a service-oriented identity management architecture', *13th Open European Summer School and IFIP TC6.6 Workshop, EUNICE 2007*, Springer, Enschede, The Netherlands, pp.1–8.

Sun, S-T. and Beznosov, K. (2012) 'The devil is in the (implementation) details: an empirical analysis of oauth sso systems', *Proceedings of the 2012 ACM Conference on Computer and Communications Security, CCS '12*, ACM, pp.378–390.

Jensen, J. (2011) 'Benefits of federated identity management - A survey from an integrated operations viewpoint', *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 6908 LNCS, pp.1–12.

Krafzig, D., Banke, K. and Slama, D. (2005) *Enterprise SOA: Service-Oriented Architecture Best Practices*, Prentice-Hall Professional Technical Reference Upper Saddle River, New Jersey.

Morgan, R.L., Cantor, S., Carmody, S., Hoehn, W. and Klingenstein, K. (2004) 'Federated security: The shibboleth approach', *Educause Quarterly*, Vol. 27, No. 4, pp.12–17.

Ragouzis, N., Hughes, J., Philpott, R., Maler, E., Madsen, P. and Scavo, T. (2008) *Security Assertion Markup Language (saml) v2. 0 Technical Overview*, OASIS Comittee Draft, 2, 2008.

Recordon, D. and Reed, D. (2006) 'Openid 2.0: a platform for user-centric identity management', *Proceedings of the 2006 Workshop on Digital Identity Management*, ACM, 3 November, Alexandria, VA, USA, pp.11–16.

Schmidt, M-T., Hutchison, B., Lambros, P. and Phippen, R. (2005) 'The enterprise service bus: making service-oriented architecture real', *IBM Systems Journal*, Vol. 44, No. 4, pp.781–797.

Sliman, L. Badr, Y., Biennier, F., Salatge, N. and Nakao, Z. (2009) *Single Sign-On Integration in a Distributed Enterprise Service Bus*, pp.1–5.

Soares, D. and Amaral, L. (2011) 'Information systems interoperability in public administration: identifying the major acting forces through a Delphi study', *Journal of Theoretical and Applied Electronic Commerce Research*, Vol. 6, pp.61–94.

Notes

[1]http://petals.ow2.org/
[2]http://servicemix.apache.org/
[3]http://www.mulesoft.org/