*Journal of Universal Computer Science, vol. 20, no. 11 (2014), 1543-1563*
*submitted: 11/11/13, accepted: 30/6/14, appeared: 28/11/14 © J.UCS*

# Middleware-Oriented Government Interoperability Frameworks: A Comparison[1]

**Giansalvatore Mecca**
(Università della Basilicata, Potenza, Italy
Giansalvatore.mecca@unibas.it)

**Michele Santomauro**
(Università della Basilicata, Potenza, Italy
michele.santomauro@gmail.com)

**Donatello Santoro**
(Università della Basilicata, Potenza, Italy
donatello.santoro@unibas.it)

**Enzo Veltri**
(Università della Basilicata, Potenza, Italy
enzo.veltri@gmail.com)

**Abstract:** We discuss deployment solutions for e-Government Interoperability Frameworks (GIFs). We concentrate on middleware-oriented GIFs, i.e., those in which middleware modules act as intermediaries among information systems that need to exchange data and services. A prominent example is the Italian SPCoop interoperability framework. We review the SPCoop architecture, and two popular open-source implementations of its core modules, called OpenSPCoop and freESBee. We argue that the comparison of these two solutions is relevant since they obey to radically different philosophies, both in terms of the relationship to the underlying J2EE container, and of their internal module organization. Then, we discuss one of the main problems in large-scale deployment of SPCoop-like GIFs, namely the need to quickly deploy a large number of middleware instances over a relatively small number of servers. We report a number of experiments to discuss how the different design choices impact performance. To the best of our knowledge, this is the first large-scale test of the framework, from which a number of important lessons can be learned.

**Keywords:** e-Government, interoperability, Government Interoperability Frameworks (GIF), middleware, domain gateways, Service Level Agreements (SLA)
**Categories:** H.4.3, H.4.2, H.3.5

## 1 Introduction

Ensuring interoperability among information systems of Public Administrations is nowadays considered a primary goal by most Governments in the World. *Interoperability* can be defined as the ability to exchange data and services by different computing systems. *Government Interoperability Frameworks* (*GIFs*)

[1] Preliminary portions of this paper appeared in [Mecca et al. 2013].

[Guijarro 2007] [Abramowicz et al. 2008] [Ibrahim and Hassan 2010] are sets of rules, guidelines, and technological standards that all agencies within one country should adopt to enable such an exchange.

In the last ten years, many countries of the world have adopted their own GIFs. These frameworks share a number of common features, the most prominent being the adoption of Web Services as the underlying technology. In fact, GIFs typically rely on *Service-Oriented Architectures (SOA)* [Krafzig et al. 2005] as a platform for remote communication and message exchange (see Section 0 for details).

In this paper, we concentrate on *middleware-oriented* GIFs, i.e., frameworks in which service providers and service consumers are decoupled by means of *middleware services*; in a middleware-oriented framework, systems do not interact directly with each other in a point-to-point fashion, but rather communicate with intermediate layers of software that are responsible for various services, among which message-format standardization, transparent routing, enhanced security, reliability, and quality-of-service monitoring.

The Italian SPCoop infrastructure [Baldoni et al. 2008] is a primary example of a middleware-oriented GIF. We focus on the SPCoop standard, review its architecture, and introduce its core module, the *domain-gateway* [Baldoni et al. 2008]. Then, we compare two open source implementation of the SPCoop standard. The first one, called freESBee [Mecca et al. 2008], has been developed by our group at University of Basilicata. The second one is called OpenSPCoop [Corradini and Flagella 2007]. We show how these two systems are based on completely different philosophies, each of which has its own advantages and disadvantages.

These notions lay the ground to discuss one of the main technical challenges towards the goal of reaching a wide-spread availability of middleware-oriented GIFs, namely the possibility of effectively deploying a large number of instances of the middleware modules over small–size server farms. In the paper, we report a number of experimental results that show how the inner organization and the architectural choices of a middleware solution deeply impact deployment policies. More specifically, we compare OpenSPCoop and freESBee in different scenarios, ranging from simple loopback ones, to more realistic point-to-point communication, with different configurations and workloads. In addition, we report the first end-to-end tests of the SPCoop framework, in which all components are in place.

Ultimately, we believe that this paper teaches important lessons to data architects that face the problem of deploying an interoperability framework.

## 2 An Overview of SPCoop

SPCoop[2] is the Italian e-government interoperability infrastructure, aimed at enabling the seamless exchange of application services among information systems of Public Administrations. As we mentioned in the previous section, it is based on a set of standards and guidelines [DigitPA], and a middleware architecture that has the following goals:

[2] SPCoop stands for “Servizio Pubblico di Cooperazione Applicativa”, i.e., public service for application interoperability.

- to standardize the format of messages, and to support the automatic enrichment of messages with standard-compliant metadata;
- to provide automatic routing, and to abstract the addressing protocol with respect to the actual location of services over the network;
- to provide advanced services, like enhanced security, identity management and single sign-on, service level agreements, and publish and subscribe.

In fact, the SPCoop standard shares a number of similarities with the goals of *Enterprise Service Buses* [Woolf 2007]. However, differently from ESBs that are typically centralized middleware components for the various information systems of a

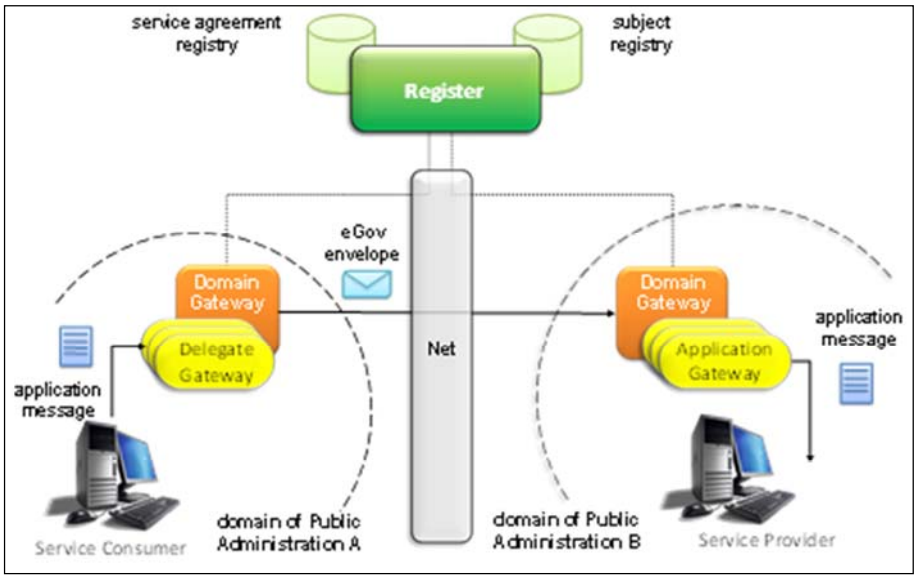


*Figure 1: The SPCoop Architecture*

complex enterprise, SPCoop is inherently distributed, since it assumes that information systems of Public Administrations are organized in independent domains that need to cooperate while maintaining their autonomy. In this respect, SPCoop can be considered as a large *distributed ESB* for the Italian e-Government.

The three main components of the SPCoop architecture are the *domain gateway*, the *e-Gov envelope*, and the *service agreement*, as shown in Figure 1. These elements are discussed in the following paragraphs.

**The Domain Gateway.** In the typical SPCoop scenario, two local information systems from different domains and networks may exchange messages through the respective domain gateways. The domain gateway is the unique access point for all information systems of a given domain to exchange services with systems of other administrations. It is composed of two main components, one for outbound and the other for inbound messages, called the *delegate gateway* and the *application gateway*

respectively.

Figure 1 describes the flow of a typical SPCoop transaction. Assume a local information system S1 from domain A intends to request services from a local information system S2 belonging to domain B. SPCoop disallows direct point-to-point communication between S1 and S2. On the contrary, an SPCoop request message originating from S1 goes first to the delegate gateway of domain A. Based on the recipient specified in the message (which we assume to be S2 in this example), the domain gateway is responsible for locating and contacting the appropriate domain gateway, the one of domain B, to transfer the message. When the message is received by the domain gateway at domain B the application gateway contacts the intended recipient S2, and delivers the message. The opposite path is followed to return the answer. In the following, we shall refer to an exchange of messages as the one depicted in Figure 1 under the term *SPCoop transaction*.

**The e-Gov Envelope.** SPCoop does not constrain the protocol and format that a local information systems should use to contact the delegate gateway, or to receive messages from the application gateway. In fact, the standard leaves some freedom in this respect to simplify the way in which legacy information systems are integrated into the platform. On the contrary, the SPCoop rules are quite strict with respect to the format of messages that domain gateways exchange with each other. These messages, called *SPCoop messages*, must adopt the SOAP 1.1 standard (with attachments), and use HTTPS as a communication protocol. In addition, the SOAP envelope must obey to a fixed grammar in terms of structure and headers that goes under the name of *e-Gov envelope*. Domain gateways are responsible of properly translating the original application messages into well-formed e-Gov envelopes, enriched with the needed headers.

**Registry Services and Service Agreements.** As it is common, registry services are provided to allow for service identification and location, security management, and quality of services. Domain gateways contact the centralized registries in order to gain information about the list of subjects that are authorized to exchange messages using the infrastructure, and the so called *service agreements*. Service agreements are XML documents stored in the appropriate registry that specify all the rules according to which service requesters and service providers are supposed to exchange services. More specifically, they describe in detail: *(i)* all subjects that are authorized requesters or providers for a given service; *(ii)* all service interfaces (in essence, fragments of WSDL code); *(iii)* a specification of the cooperation profile for each service, either synchronous, asynchronous, or one-way. In addition, they also provide information about infrastructure services, namely security policies for the service and service level agreements. These are described in more detail in the next section.

### 2.1 Evolutions of the Standard: the ICAR Project

SPCoop was conceived as a nation-wide effort, and therefore aims at supporting interoperability among a very large number of subjects. Besides central Public Administrations, these include also local Public Administrations, like Regions and municipalities, hospitals, and other local agencies.

To alleviate the problem of deploying the framework within local administrations, it was natural to explore hosting solutions in which larger

administrations acted as technology facilitators for smaller ones, offering hosting services for their domain gateways. To give an example, Regions were soon identified as natural intermediaries for smaller municipalities within their territories.

In 2006, the ICAR initiative [CISIS] was started by 18 Italian regions in order to investigate evolutions of the standard in which the role of regions as facilitators was more prominent. The ICAR project brought two main additions to the SPCoop standard, as follows (see also Figure 2): *(i)* it introduced a notion of a *network*

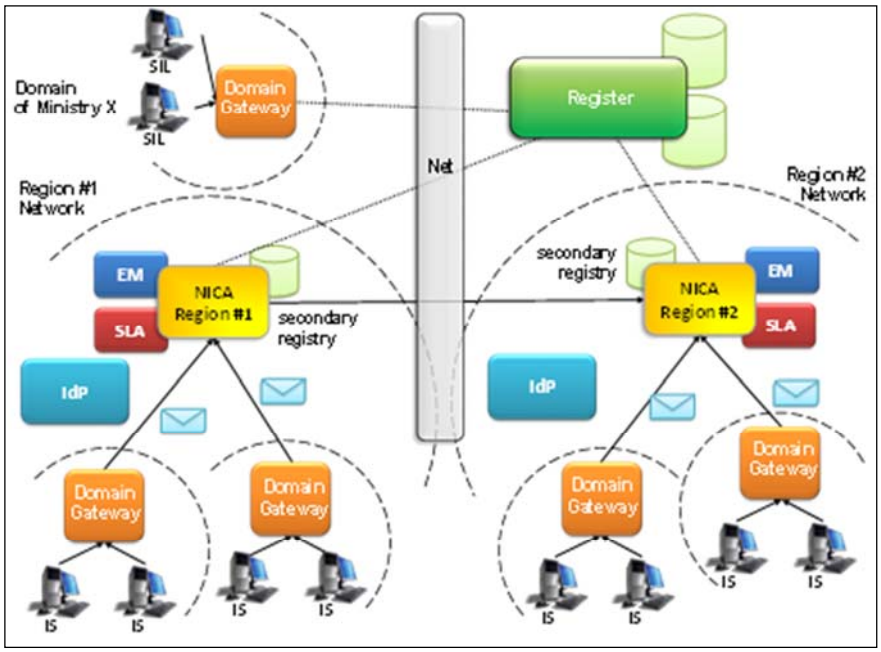


*Figure 2: The ICAR Evolution*

*interoperability node* (NICA) as a central hub for a regional domain; *(ii)* it explored a number of *infrastructure services* that were in part described in the standard, but needed to be made more concrete, namely: *federated identity*, *service-level agreements* (*SLA*), and *event management*. We discuss some of these in the following paragraphs.

**Network Interoperability Nodes (NICAs).** The organization of a SPCoop network with NICAs is depicted in Figure 2. As it can be seen, SPCoop domains of central administrations remain essentially untouched. On the other side, interoperability domains of local administrations are organized within *regional networks*. Each regional network is orchestrated by a NICA that plays the role of a mediator among domain gateways. In essence, NICAs provide advanced routing facilities to domain gateways within the network, in the sense that a domain gateway only needs to know the endpoint of the regional NICA; whenever an information system within the local domain requests a service, the domain gateway forwards the e-Gov envelope to the NICA. Each NICA knows every other NICA, and the domain gateways of central

administrations, and takes care of properly routing the message. To this end, it incorporates a secondary registry, which is transparently synchronized with the primary registry. In this way the configuration and management of domain gateways within a regional domain is highly simplified. In addition, NICAs also centralize a number of value-added services described in the following paragraphs.

**Service Level Agreements (SLAs).** The first, important service is the management of *service-level agreements* (SLAs). Service providers are supposed to guarantee minimal levels of services to their clients. These levels are explicitly stated within service agreements, and their respect must be monitored in a continuous way. Each SLA definition is composed by a parameter and a threshold value. Examples of parameters are average response time, maximum transaction time, percentage of errors per day, and so on. For all SPCoop transactions, service providers have the responsibility of recording the value of any parameter that is mentioned in a service agreement within the SLA manager, a module of the infrastructure that logs parameter values, and offers query and analysis functionalities. Domain administrators may periodically query the SLA manager module in order to detect violations to SLAs or simply explore bottlenecks and possible optimizations. NICAs offer support for the management of SLAs to information systems that belong to the regional network by providing a centralized SLA manager for the entire regional network.

**Other Modules.** Here we briefly mention a few other modules introduced by the ICAR project, namely *Event Managers* (EMs), and *Federated Identity Management* (IM). An in-depth discussion of these modules falls outside of the scope of this paper, and for that we refer the reader to a companion paper [Mecca et al. 2014].

Event Managers implement *event-driven* communication protocols [Taylor et al. 2009]. This kind of communication proves very helpful in all cases in which many subscribers need to be notified by a few publishers about changes in the state of things. In these cases, it is much more effective to decouple publishers from subscribers through a *publish and subscribe* infrastructure.

Identity Providers are used to store user credentials and to maintain assertions about user *attributes*, i.e., descriptions of the roles that a user may play which authorizations are based upon. A natural requirements is that information systems that belong to interoperability domains provide *single sign-on* capabilities. This means that a user is required to authenticate once, and then her/his credentials and attributes are stored for the length of the session in such a way that s/he does not need to provide them for subsequent accesses, as discussed in [Mecca et al. 2014].

## 3 Open-Source Domain Gateways: OpenSPCoop

In this section and the following we discuss two certified open-source implementations of the SPCoop domain gateway. We start with OpenSPCoop, and then introduce freESBee in the next section. While the SPCoop specification does not prescribe any development platform, Java 2 Enterprise Edition is a de-facto standard in this field. In fact, both implementations adopt J2EE.

OpenSPCoop[3] has been historically the first open-source implementation of the SPCoop domain gateway. It offers a suite of modules: an implementation of the domain gateway, a registry service, and an event manager to support publish and subscribe applications. In fact, the OpenSPCoop codebase was also used as a basis to develop the ICAR NICA reference implementation. In the following, we shall refer to the original plain domain gateway simply as "OpenSPCoop", and to the reference implementation of the NICA as the "ICAR NICA".

**The Container-Bound Architecture of OpenSPCoop**. OpenSPCoop[4] is a J2EE Web application developed for the JBoss application server[5]. To understand the relationship between OpenSPCoop and JBoss it is useful to recall that domain gateways are message-oriented applications, i.e., they listen on appropriate network endpoints for incoming messages, receive the messages, process them, and then forward the result to the destination network endpoint. In this respect, a crucial component of the overall application is the adoption of an appropriate *messaging service*, i.e., an infrastructure to receive, store, process, and exchange messages among components.

OpenSPCoop relies on the *Java Message Service* (*JMS*) [The Java Community Process] specification, one of the messaging technologies developed as part of the Java platform, and more specifically on its implementation as part of the JBoss application server. In essence, a JMS implementation is a set of queues, each with an appropriate network endpoint, which can be used to receive, store, and pull out messages that have been sent to the associated endpoint. An OpenSPCoop domain gateway is deployed as a Web application within a JBoss installation. As part of the deployment, users need to specify the endpoints of the JMS queues that OpenSPCoop will use as an underlying infrastructure to process SPCoop messages. In this respect, the OpenSPCoop-JBoss integration is quite tight, so that we may say that OpenSPCoop adopts a *container-bound architecture*.

This choice has advantages and disadvantages. On one side, it somehow reduces portability and generates a limited form of platform lock-in, since it is not easy to deploy an OpenSPCoop gateway outside of JBoss. On the other side, it guarantees very good performance, given the strong integration with the J2EE container and the excellent JMS implementation provided as part of JBoss.

**The Tight-Integration Architecture of the ICAR NICA**. The reference implementation of the NICA system was developed starting from the OpenSPCoop codebase. It offers a SLA management module and an event manager, both developed as internal modules of the NICA system and accessible through Web services hosted by the NICA. The SLA management system exposes a Web service in order to log parameter values, and an interface for queries and analysis. Similarly, the Event Manager exposes Web services to subscribe and publish events. Notice how the level of integration between the two modules and the NICA is quite tight, since they share the same database.

---

[3] http://www.openspcoop.org

[4] The authors have recently published a beta version of OpenSPCoop 2.0, which will be compatible with Tomcat as well. Given the preliminary nature of this new version, in the paper we decided to stick to OpenSPCoop 1.x.

[5] Now Wildfly, available at http://www.jboss.org.

## 4 freESBee

freESBee[6] is an alternative implementation of the SPCoop standard that was developed at University of Basilicata with the goal of exploring the boundaries of the standard, and to help clarifying some of its features. In fact, the development of freESBee originated by two main observations. First, despite the consolidated nature of the SPCoop standard, the technical documents describe the domain gateway essentially by a set of use-cases, and there is no shared conceptualization of the internal architecture and behaviour of the gateway. Second, while the technical specification is rather prescriptive with respect to the core components, there are other components of the middleware – primarily the SLA management service and the event manager – for which the specification leaves wide margins of freedom, to the point that the few existing implementations are often quite ad-hoc.

The freESBee project provides a number of modules that cover all functional aspects of the SPCoop platform. In fact, the freESBee suite is composed of the following sub-project:

- *freESBee* itself is the domain gateway, and offers support for registry services; in addition, freESBee is also a fully featured NICA;
- *freESBee-SLA* is the service-level agreement management module; it interacts with the domain gateway to log all metadata that are needed in order to monitor quality of service;
- *freESBee-GE* is the event manager;
- *freESBee-SP* is the identity management module; it is conceived to act as a middleware layer between the local information system and the domain gateway on one end, and the identity/attribute provider – like, for example, Shibboleth – used to authenticate and authorize message exchanges in the SPCoop domain.

The main architectural choices behind these modules are described in the next paragraphs.

**Enterprise Integration Patterns**. One of the main contributions of freESBee consists in its design of the internals of the domain gateway, which builds on a well-known conceptual model for messaging services, namely *Enterprise Integration Patterns* (*EIPs*). EIPs [Hohpe and Woolf 2004] are an application of the design pattern approach to messaging services. They allow one to describe the processing and pathways of a message within a system in terms of a few elementary components, like “channels”, “endpoint”, “router”, “message enricher” and so on. For a detailed description please see www.enterpriseintegrationpatterns.com.

Differently from other pattern languages, EIPs have the nice feature that is it typically possible to design or describe a complete message-oriented application simply by means of the composition of a number of patterns. In this way, they provide an elegant formalism to conceptualize and document the internals of a complex software system, leaving alone the benefits in terms of development and maintenance.

There are in fact, various open-source libraries that support the development of EIP-based applications. These libraries provide ready-made building blocks that an

[6] http://freesbee.unibas.it/.

application developer can easily customize and compose. The overall architecture of the SPCoop domain gateway in terms of EIPs is shown in Figure 3. freESBee is the result of implementing this architecture on top of Apache Camel[7], a well-known implementation of EIPs.

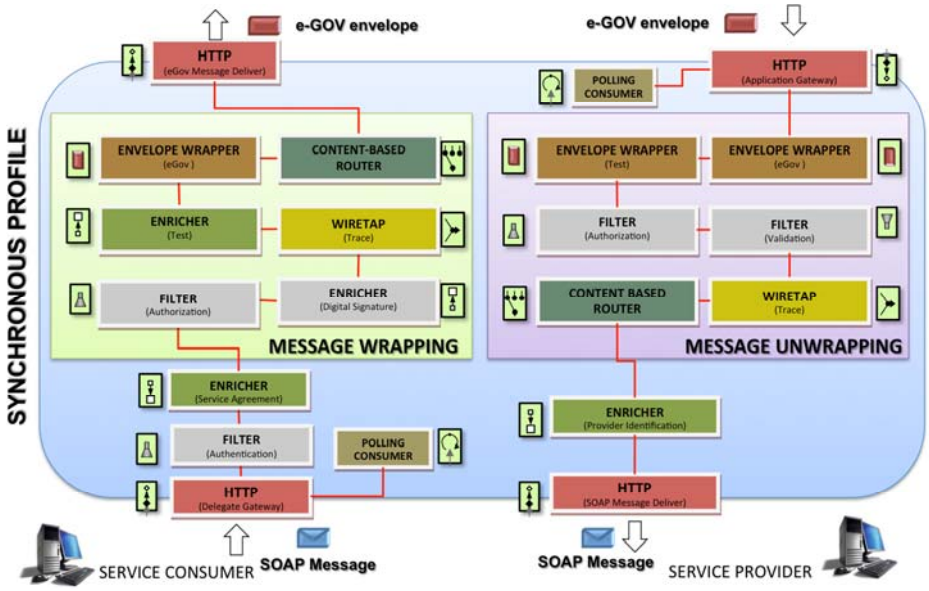


*Figure 3: SPCoop Gateway in terms of EIPs*

**A Container-Independent Architecture**. A second, major advancement made by freESBee is its *container-independent architecture*. In fact, the EIP-based design, and the Camel-based implementation are such that the freESBee domain gateway does not rely in any way on the messaging services of the underlying J2EE.

Guaranteeing this independence has been a precise design guideline throughout the development of freESBee. In fact, the software does not use JMS in any way, neither any of the endpoints offered by the J2EE container. On the contrary, each freESBee installation starts its own set of endpoints for delegate and application gateways, based on a Jetty[8] internal engine. This has two main consequences: *(i)* each freESBee instance is fully independent from the container, since it does not rely on any of its services; *(ii)* freESBee can be freely used in conjunction with any of the major J2EE containers (Apache Tomcat, JBoss, Glassfish); in fact, its level of independence is such that with minimal effort freESBee can also be deployed as a stand-alone application outside of any J2EE container, with benefits in terms of build time, especially during the development phase.

[7] http://camel.apache.org/
[8] Jetty is a well-known open-source http server available at http://www.eclipse.org/jetty/.

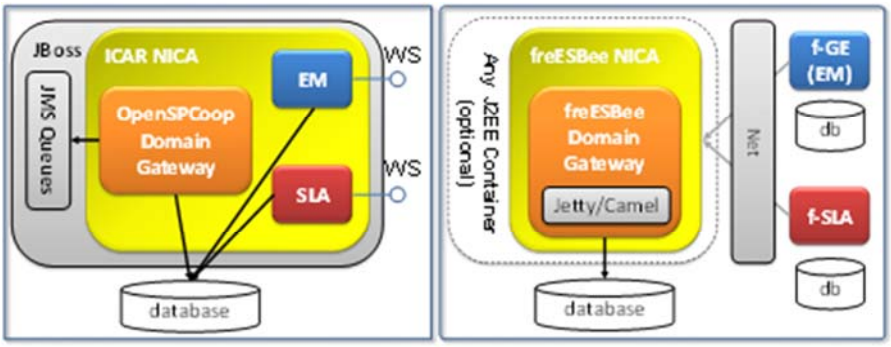


*Figure 4: Comparison between the two architectures*

**The freESBee NICA and the Loose-Integration Approach**. The same philosophy also inspired the architecture of the freESBee NICA. When designing the modules of the NICA system, we decided to follow a more lightweight approach in comparison to the one that has been followed by the ICAR/OpenSPCoop implementation. We realized that, once the middleware architecture is in place, event management and SLA monitoring do not need to be provided as ad-hoc services, but can rather be ordinary SPCoop services with their own service agreements, and therefore can be used through standard SPCoop transactions – like accessing citizenship information or recording medical treatments. There are several advantages that come with this solution:

- first, the inner architecture of the NICA module is strongly simplified, since it is stripped off of sub-modules that do not serve the purpose of routing messages; in fact, with this solution, a NICA is little more than a domain gateway, empowered with more sophisticated routing facilities;
- second, the level of coupling among the NICA system, the event manager and the SLA monitoring system decreases significantly; this is important to simplify the evolution of the various modules (especially since SLA management is still in its infancy within the framework); also, it reduces bottlenecks due to shared resources, like the database;
- third, there are significant advantages in terms of security associated with the choice of making the event manager and the SLA monitoring system standard SPCoop systems; the most prominent one is the fact that it is now possible to reuse the federated identity architecture and the single-sign on features of the platform to protect the endpoints, while in the tight-integration solution adopted by the ICAR NICA these need to be protected through completely different policies.

In summary, a comparison between the two solutions is reported in Figure 4. On the left hand side of the figure, we see an abstraction of the ICAR/OpenSPCoop solution. Notice the container-bound relationship to the JBoss application server, and the tight integration with the event manager and SLA modules. On the right hand side is a

representation of the freESBee solution. Here, there is no dependence at all with respect to the J2EE application container. Also, freESBee-GE (the event management module), and freESBee-SLA have been externalized and turned into standard SPCoop service providers, that interact with the rest of the subjects in the network through domain gateways and SPCoop envelopes.

Notice, however, that the two solutions are completely interoperable with one another, since they are based on the same standard. In other terms, it is possible to build mixed architectures in which a freESBee domain gateway interacts with an ICAR NICA, or an OpenSPCoop gateway with a freESBee NICA.

## 5 Massive Deployment of Domain Gateways

Deploying the SPCoop framework is currently an ongoing effort in Italy. A study dating back to 2010[9] reports over 417 installations of the SPCoop domain gateway in the country. While we expect this number to have increased in the last few years, we are still far from the goal of having a domain gateway for each public administration.

As we mentioned, one of the primary concerns is the need to deploy the framework not only at the central level, but also at the level of local administrations. Since each of these administrations typically represents an application domain by itself, and each domain needs a domain gateway in order to interoperate with other parties, a complete deployment of the framework requires to deploy a large number of middleware instances.

It was soon realized that only a few administrations have the skills and resources to manage autonomously their SPCoop platform, especially at the local level. On the contrary, for many smaller realities the actual management of the middleware platform turned out to be a major technological and organizational problem. As a consequence, IT departments of Italian Regions were requested to host within their server farms hundreds of different instances of SPCoop domain gateways.

The straightforward solution to this problem consists in having a dedicate server for each domain gateway instance. We call this the *dedicate-server* architecture; this is closer to a housing solution, and guarantees the highest level of independence for the local administration, since its SPCoop transactions are handled by a separate host; however, when the number of instances is in the order of the thousands[10], this solution becomes clearly unfeasible because of the number of different servers that are needed.

To mitigate this problem, one may think to resort to cloud-based virtualization solutions. Unfortunately, virtual machines are of little help in this framework, for two main reasons. First, the current regulations in terms of data security and storage for Public Services in many European countries do not allow yet to adopt external, cloud-based solutions [Rebollo et al. 2012]. Second, and more important, even in those cases in which this is possible, the *total cost of ownership (TCO)* [Ellram 1993] of thousands of independent servers, either physical or virtual, is unacceptable due to the need to configure and administer each one independently from the others.

[9] Source: http://www.riir.it/sites/default/files/RIIR_2010_light.pdf

[10] In Italy there are several Regions of this size. To give an example, Region Lombardia includes 1.544 municipalities.

When this was realized, a proposal was put forward to create *multi-subject domain gateways*. A multi-subject domain gateway is a single instance of the gateway that does not serve a single administration, but possibly more than one. In essence, a multi-subject domain gateway for administration A (e.g., a hospital) and administration B (e.g., a local municipality) encapsulates all information systems of administration A and all information systems of administration B, as if they belong to the same interoperability domain. This solution is easily realized, since it amounts to adding a few more endpoints to the domain gateway. However, this is hardly satisfactory from the functional viewpoint. In fact, it does not guarantee any form of independence between the two domains, especially in terms of security. Being handled by the same domain gateway, all information systems of A and B share the same database, logs, message infrastructure and bandwidth. For this reason it might be delicate in many cases to handle failures, security breaches, and poor performance. Since the overall point of a middleware-oriented GIF is to preserve the autonomy and independence of the local information systems, in the following, we shall not investigate further this solution.

On the contrary, in this paper we investigate solutions to the problem of hosting multiple independent instances of a domain gateway within a single server. There are two main strategies to do this.

(a) the *single-container strategy*, according to which a single server offers a single J2EE container, and all instances are deployed as Web applications within the same container; based on what we discussed so far, it should be easy to see that this is possible only if the gateway implementation adopts a container-independent architecture, otherwise the different instances would collide with each other when using the messaging service infrastructure offered by the container (messaging queues and endpoints);
(b) the *multiple-container* strategy, in which several instances of the J2EE container are installed within a single server, i.e., several JBoss or Tomcat installations listening on different ports of the same server, each with a single instance of the domain gateway; this is the only viable option for container-bound gateways like OpenSPCoop.

We therefore have three possible solutions to compare: *(i)* the dedicate-server solution, with one server per domain gateway; *(ii)* the single-container solution, with multiple domain gateways per server within a single J2EE container; *(iii)* the multiple-container solution, with multiple domain gateways per server, each within a different J2EE container.

Before we turn to our experimental comparison in the next section, we want to emphasize that the three solutions have completely different TCOs. Notice that a complete estimate of the actual TCO of a deployment is not possible here, since there are many technical parameters that may vary (e.g., configuration and cost of the physical/virtual machines, skills and cost of the personnel, cost of utilities et cetera). However, a few considerations are possible.

First, under the same values of the technical parameters, the TCO of the dedicate-server solution is significantly higher than those of the shared-server ones, because of the following components: *(i)* need of a larger server-farm due to the considerably higher number of servers, with increased cost in terms of space and utilities; *(ii)*

higher cost of the hardware; *(iii)* higher configuration and maintenance costs of the servers; *(iv)* higher configuration and maintenance cost of the network configuration.

Based on these observations, in the following we disregard this solution, and concentrate on the other two. Between these, we notice that under the same technical parameters, the multiple-container solution again has a higher TCO with respect to the single container solution. This is due to the cost of configuring and administering the higher number of independent instances of the J2EE container.

Given this preliminary observation, in the next Section we compare the performances of these two solutions, and gain deeper insights about their respective advantages.

## 6 Experiments

We conducted a number of experiments using both freESBee and OpenSPCoop. The main goals were to test the scalability of the two solutions, under two main perspectives. On one side, we measure the throughput achieved by the domain gateway, i.e., the average number of messages per second handled during a test session. On the other side, we are interested in investigating how the throughput varies for the two solutions when the number of instances of the domain gateway per server increases.

For the purpose of our tests we set-up three different interoperability scenarios, as shown in Figure 5.

**Scenario a: Loopback.** This is a simple loopback scenario, with one service provider, one domain gateway, and a number of service requesters. This scenario was conceived to test the scalability of the domain gateway in isolation. Since we are not interested in testing the actual performance of the service provider, nor that of the service requesters, but rather the one of the domain gateway, we chose as a service provider a simple “echo” Web service developed for this test, with messages of 1Kbyte each. The service provider runs on the same physical machine as the domain gateway. To simulate the traffic originated by an increasing number of clients, we used Apache JMeter v. 2.7. JMeter was configured to simulate variable numbers of concurrent clients, ranging from 10 to 100, each one issuing repeated requests to the service provider through the domain gateway, for a total of 10.000 messages sent for each test session. All components run on the same machine, which was deliberately chosen with a high-end profile (a MacBook Pro with a quad-core Intel i7 running at 2.6 GHz, a 512 GB solid-state HD, and 8 GB of RAM).

The DBMS used by the domain gateways to log transactions was PostgreSQL 9.1. We ran scalability tests both for single-instance-per-server and multiple-instance-per-server configurations of the middleware. Proper care was needed in multiple-instance tests in order to prevent that the DBMS connection pool was saturated. Since PostgreSQL accepts a maximum of 120 concurrent connections, to execute a test with X instances of the domain gateway we configured each instance to ask for a maximum of 120/X connections.

**Scenario b: Point-to-Point.** This is a more typical point-to-point interoperability scenario, with two domains: a number of service requesters (simulated using JMeter)

run on one domain, with an appropriate domain gateway, and send messages to a service provider that runs in a different domain, with a different domain gateway. The service provider implements a real-life service – it returns the set of cars owned by a person, given its unique national identifier (the taxpayer number in Italy). Request messages contain a random identifier, chosen by a pool that was configured into JMeter, and have size between 7 and 8KB. This is a rather typical size for this kind of messages when WS-Security is used to crypt and digitally sign the payload. The service provider runs an actual query on the database to retrieve the answer.

In this second case, our priority was to be close to the actual implementation of these services in local administrations. Therefore, we chose lower-end machines. Both the server and the client are dual-core machines with an Intel Xeon Processor at 3GHz, 7200 rpm SATA disks and 8GB of RAM. The operating system was Lubuntu 12.10 and the DBMS was PostgreSQL. Also in this case we ran scalability tests both with single instances and multiple instances per server, and used these to compare the two implementations of the domain gateways.

**Scenario c: Full Architecture.** This is a NICA-oriented scenario, in which all components of the framework are in place. As it can be seen in Figure 5, this requires to deploy a number of components that is significantly higher than in previous scenarios, including the event manager, the SLA management module, and an IdP to provide federated identity services. Also this test was run on the servers used for scenario a.

To the best of our knowledge, this is the first end-to-end extensive test of the SPCoop/ICAR infrastructure, and therefore we believe it has its merits. The use-case we adopted is event-oriented, and is referred to public auctions. Events correspond to the opening and closing of auctions of public goods. Clients that subscribe an auction receive notifications about all changes of state of a good, and may make offers. In this scenario, it makes little sense to measure the scalability of one or the other component. We believe it is more interesting to use SLA to investigate how an increased workload impacts the SLA parameters registered by service providers, primarily the average and maximum response time.

**Test Configurations.** In all scenarios, we tested the most recent stable version of OpenSPCoop (v. 1.4.1) available on the Web site, configured according to the standard set of configuration parameters suggested on http://openspcoop.org under JBoss. As suggested by the Web site, we used JBoss v. 4.2.3, although this is not the most recent version of JBoss available at the moment. Due to its container-bound architecture, multiple-instance tests were run by installing several instances of JBoss on the server, each with a single copy of OpenSPCoop. For some of our experiments, we also ran additional tests using the new version of OpenSPCoop (v. 2.0beta2) under Tomcat.

We also tested the latest available version of freESBee (v. 2.2), deployed according to the standard set of configuration parameters suggested on http://freesbee.unibas.it. We selected Tomcat v. 7.0.47 as a container for freESBee, due to its wide adoption and ease of configuration. Multiple-instance tests were run by deploying different copies of freESBee inside the same Tomcat instance.

Since we want to measure the performance of the middleware modules, in all tests we connected the various components using a dedicated high-speed Gigabit Ethernet networks. In this way, network latency was negligible, no other traffic impacted our tests results, and the network never became the bottleneck.

Throughput values in our results are the ones reported by JMeter in its Summary Report (average number of requests completed per second during a test session). In addition, we used the machine built-in profiler to measure RAM occupation and CPU workload. We repeated each experiment 5 times, and took the average result for each output value. A test was successful if we were able to complete the expected number of messages with less that 1% of errors, and none of the domain gateway instances stalled. It was considered as failed otherwise. The two main causes of failures were out-of-memory errors on the server side, and inability to obtain connections within a given timeout from the DBMS.

**Test Results – Scenario a: Loopback.** Let us begin with Scenario a., the loopback scenario. Recall that this scenario was primarily conceived to test the scalability of the domain gateway in isolation. We ran both single-instance and multi-instance tests. Figure 6.a.1 reports scalability results for single-instance test. In this case, a single instance of the domain gateway was installed, and throughput was recorded for increasing numbers of concurrent clients, ranging from 10 to 100. It can be seen that OpenSPCoop v.1.4.1 under JBoss reaches a considerable throughput of 120 messages

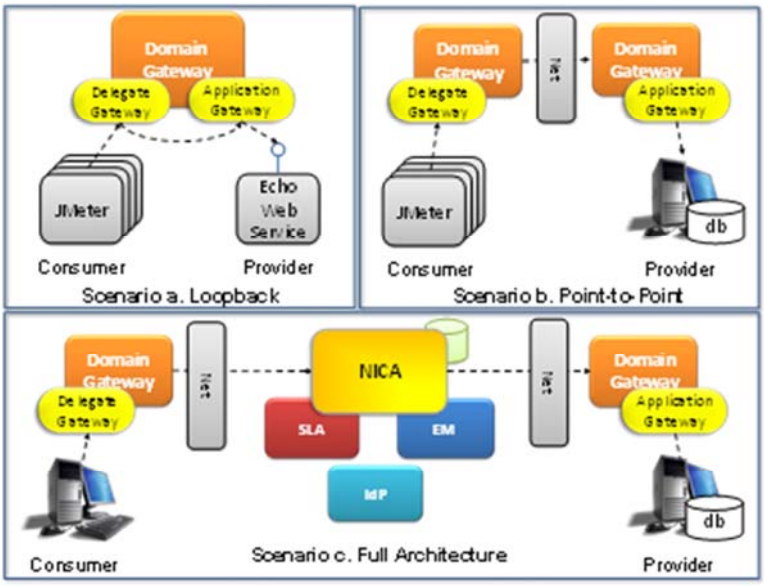


*Figure 5: Interoperability scenarios used in experiments*

per second for 10-20 concurrent clients. Then, performance decreases as soon as the PostgreSQL connection pool becomes the bottleneck. We ran the same tests also using OpenSPCoop v.2.0beta2 under Tomcat to confirm the results, and no noticeable change of performance was recorded.

In previous tests reported in [Mecca et al. 2013], the throughput of freESBee was slightly lower than OpenSPCoop's. This was due to the fact that the version of freESBee we used at the time (v. 2.1) did not use any form of main-memory caching. More specifically, whenever the domain gateway needed to route a message, and therefore access the details of the service agreement for that SPCoop transaction, it ran a query over the local database to fetch the needed records. Notice, however, that these queries are repeated for each message, so that they may introduce a considerable overhead.

In version 2.2 we introduced a main-memory cache for service agreements. In the new version, service agreements are fetched from the database at the first message, and then stored for a fixed time-to-leave in the cache. In Figure 6.a we report throughputs for both the execution of freESBee with and without caching. It can be seen that enabling the cache brings considerable benefits in terms of scalability, to the point that freESBee outperforms OpenSPCoop, reaching a maximum of approximately 150 messages per second.

Figure 6.a.2 and Figure 6.a.3 report results for multi-instance tests on the loopback scenario. As it was expected, freESBee had better performance in multi-instance tests. Figure 6.a.3 reports RAM occupation for multiple-instance tests ranging from 2 to 20 domain gateway instances per server. Tests were run for 20 concurrent clients per instance, i.e., the 20-instance test with 10 clients per instance simulated the load of 200 concurrent clients. It can be seen that, memory-wise, the container-independent architecture of freESBee guarantees better performance, so that it is possible to run up to 20 instances of the domain gateway inside a single server.

On the contrary, several of the OpenSPCoop tests failed. In the 20-client configuration, the maximum number of OpenSPCoop instances for which the test succeeded was 8. Above those numbers some of the instances stall, and high numbers of errors are observed. This is due to the nasty interaction between excessive memory usage and aggressive requests for connections by the multiple JBoss instances to the DBMS.

Figure 6.a.2 report throughput results for multiple-instance tests. It can be seen how the throughput of freESBee improved as the number of instance increased, as long as the CPU of the server machine was not saturated. In fact, given the less aggressive thread management policy of Camel, the presence of multiple instances, and therefore of multiple Camel contexts, brings an increase in parallelism.

**Test Results – Scenario b: Point-to-Point.** Differently from the loopback scenario, the second, point-to-point test scenario is more faithful to the typical execution of SPCoop transactions. We expect lower throughputs with respect to the ones reported for scenario a, for several reasons. First, recall that the server configuration we chose in this case is relatively low-end. We believe this is quite close to what typically happens in the server farms of most public administrations in Italy. Second, the SPCoop transaction now starts at the requester's domain gateway, and crosses the network to reach the provider's domain gateway.

Overall, Figure 6.b.1 and Figure 6.b.2 confirm the results discussed above. Also in this case freESBee (with caching) guaranteed a higher throughput, and it was less eager in terms of memory consumption due to the container-independent, multiple-gateway per container configuration. OpenSPCoop was unable to complete the test for a number of instances higher than 8.

To summarize, our experiments show that the container-independent architecture of freESBee may achieve better performance in SPCoop installations, especially when the number of instances per server increases.

**Test Results – Scenario c: Full Architecture.** Figure 6.c reports results for our final test, based on Scenario c. Here, we deployed all components of the framework. Besides the domain gateways, we also put in place: (a) a regional NICA; (b) an event manager; (c) a SLA management module; (d) an IdP to handle security and authorizations based on single-sing-on. This test was performed only using the freESBee platform, since we do not have access to the ICAR NICA. In fact, to the best of our knowledge, this is the first end-to-end experiment of the SPCoop framework.

Notice that transactions become so involved, in this case, that measuring scalability in terms of messages per second, or even RAM usage at one or more of the nodes makes little sense. In other terms, the deployment is too complex to justify unit tests, like the ones we performed in previous scenarios. As a consequence, we decide to perform a functional test, and to do this, we rely on the SLA management architecture. More specifically, we report three SLA parameters related to transactions, and measured at the service consumer, namely minimum, maximum, and average transaction time.

The most prominent evidence of this test is that exercising the framework in its entirety brings to execution times that are considerably higher than in the previous scenarios. Recall from Figure 6 that in the loopback case on a high-end computer we were able to process hundreds of transactions per seconds. In the point-to-point case – stripped down of any additional processing related to SLAs, event management and security – the throughput was lowered to dozens of messages per second, which is still pretty high, especially considering that the servers were mid-range in this case.

As soon as we put in place all of the components, transaction times rise to a few seconds per transaction, i.e., less than a message per second. This is not surprising, considering the overhead associated with the routing throughout the two domain gateways and then the NICA, queries to the IdP, and then SLA recording.

This should help to put in perspective the adoption of such an elaborated architecture: even middleware modules that in isolation have excellent scalability will hardly scale to large throughputs due to the complexities of the framework.

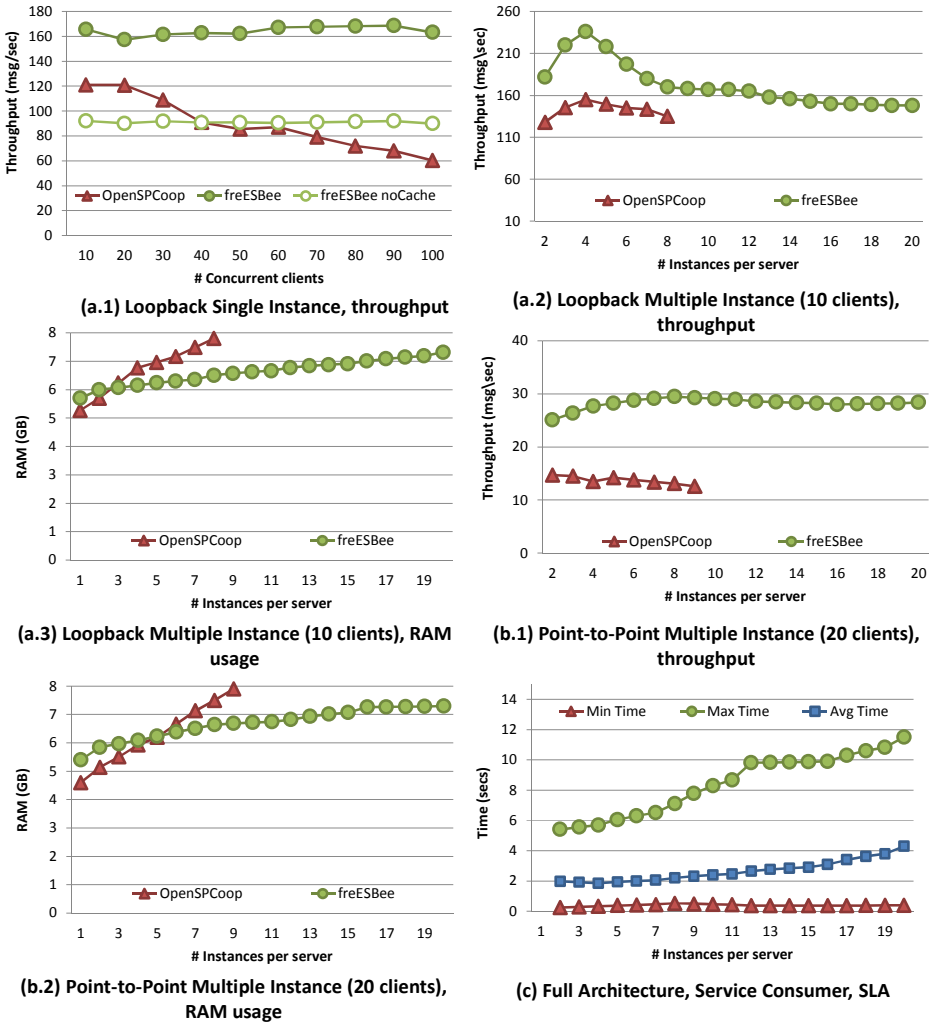


*Figure 6: Test Results*

## 6 Related Work

In the last ten years, many countries of the world have adopted their own GIFs. These have been the subject of several recent surveys [Guijarro 2007] [Abramowicz et al. 2008]. Here we discuss a few aspects. Broadly speaking, we may classify SOA-oriented GIFs in two main categories.

On one side we have *lightweight* GIFs; these frameworks adopt the standard service-oriented point-to-point architecture, in which a *service consumer* that needs to access data or applications, and a *service provider* that offers access to data or

applications, directly exchange messages with one another through Web services organized according to a common format. These GIFs consist essentially of collections of technological standards and guidelines that consumer and provider systems should adhere to in order to participate in interoperability initiatives. Lightweight GIFs represent the majority in Europe [Guijarro 2007]. Main examples of lightweight GIFs are the UK e-GIF framework [The UK e-Government Unit], the French CCI [ADAE], and Germany's SAGA framework [Federal Government Co-ordination and Advisory Agency for IT]. At different degrees, all of these frameworks are primarily collections of technical documents that provide IT specialists with references to standards and guidelines in order to build interoperable solutions within e-Government services.

A similar solution has been adopted by the USA within their Federal Enterprise Architecture Framework (FEAF) [Federal Chief Information Officers Council]. The FEAF initiative, however, departs from the ones in Europe primarily because of the federal nature of the country.

On the other side, we have *heavyweight* GIFs; these frameworks typically adopt a more involved architecture, in which service providers and service consumers are decoupled by means of middleware services. This is the category we focus on in this paper. We have discussed the Italian SPCoop infrastructure [Baldoni et al. 2008] in detail in the paper. Another example of this kind of architecture is represented by the early efforts towards a European Interoperability Standard within the IDA eLink initiative [European Dynamics 2004]. The eLink architecture is virtually identical to the SPCoop one, since it envisions application domains within the various European countries, decoupled by domain gateways (the eLink gateway). More recently, European efforts towards a general interoperability framework for e-Government services have evolved into the EIF initiative [IDABC]. While EIF is a more ambitious and higher level framework, domain gateways are still present under the name of *interoperability facilitators*.

More recently, e-Government frameworks have progressed with the goal of incorporating forms of semantic interoperability, in the spirit of the Semantic Web [Sabucedo and Anido Rifòn 2010]. Semantic interoperability allow services to share a common vocabulary, and facilitates the integration of different domains.

## 7 Conclusions

Government Interoperability Frameworks are essential initiatives in order to promote the adoption of advanced platform for Government-to-Government and Government-to-Citizen services. Among these, middleware-oriented GIFs pose a number of technical challenges, primarily related to the deployment of the middleware modules.

In this paper, we reported an in-depth analysis of the Italian SPCoop framework. We reviewed its main components, and investigated the two main open-source implementations. We showed that the internal and external design of an implementation may deeply impact its performance in different deployment scenarios. Our experiments show that the container-independent architecture of freESBee may achieve better performance in SPCoop installations, both in terms of throughput and memory occupation. This is especially true when the number of instances per server

increases, and may contribute to lower the TCO of freESBee-based deployment solution with respect to the alternatives explored in the paper.

In addition, we report the first comprehensive end-to-end tests of the SPCoop infrastructure. These tests show that complex transactions that need to traverse multiple agents throughout the network have performances that may significantly differ from those of simple messages exchange. This sheds some further light on the distinction between lightweight and heavyweight GIFs. We believe these results may be of help to middleware developers and system managers, and represent a useful reference for data architects that face the need to deploy e-Government services. In fact, we consider this paper a first important step towards more systematic studies of middleware modules in e-Government architectures.

## References

[Abramowicz et al. 2008] Abramowicz W., Bassara A., Wisniewski M. and Zebrowski P. - Interoperability Governance for e-Government. In: Information Systems and e-Business Technologies. Berlin Heidelberg: Springer-Verlag; 2008. p. 14 - 24.

[ADAE] Le Cadre Commun d'Intéroperabilité. [Internet]. http://www.adae.gouv.fr.

[Baldoni et al. 2008] Baldoni R., Fuligni S., Mecella M. and Tortorelli F. - The Italian e-Government Enterprise Architecture: A Comprehensive Introduction with Focus on the SLA Issue. In: *Proceedings of the 5th International Service Availability Symposium, ISAS 2008*; 2008; Tokyo, Japan.

[CISIS] The ICAR Project Web Site. [Internet]. http://www.progettoicar.it.

[Corradini and Flagella 2007] Corradini A. and Flagella T. - OpenSPCoop: un Progetto Open Source per la Coperazione Applicativa nella Pubblica Amministrazione. In: *Atti del Convegno Italiano AICA*; 2007.

[DigitPA] Servizi di Interoperabilità Evoluta. [Internet]. http://www.digitpa.gov.it/spc/servizi-interoperabilit-evoluta.

[Ellram 1993] Ellram L. - Total Cost of Ownership: Elements and Implementation. International Journal of Purchasing and Materials Management. 1993;29:2-11.

[European Dynamics 2004] IDA eLink Specification. [Internet]. 2004 http://ec.europa.eu/idabc/servlets/Doc1a78.pdf?id=18685.

[Federal Chief Information Officers Council] Federal Enterprise Architecture Framework (FEAF). [Internet]. http://www.whitehouse.gov/omb/e-gov/fea.

[Federal Government Co-ordination and Advisory Agency for IT] SAGA. [Internet]. http://www.cio.bund.de/Web/DE/Architekturen-und-Standards/SAGA/saga_node.html.

[Guijarro 2007] Guijarro L. - Interoperability Framewors and Enterprise Architectures in e-Government Initiatives in Europe and the Unites States. Government Information Quarterly. 2007;24:89-101.

[Hohpe and Woolf 2004] Hohpe G. and Woolf B. - Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions. Prentice Hall; 2004.

[Ibrahim and Hassan 2010] Ibrahim N.M. and Hassan M.F. - A survey on different interoperability frameworks of SOA systems towards seamless interoperability. In: *International Symposium on Information Technology (ITSim 2010)*; 2010. p. 1119-1123.

[IDABC] EIF - European Interoperability Framework for pan-European eGovernment services. [Internet]. http://ec.europa.eu/idabc/en/document/2319/5644.html.

[Krafzig et al. 2005] Krafzig D., Banke K. and Slama D. - Enterprise SOA: Service-Oriented Architecture Best Practices. Prentice Hall; 2005.

[Mecca et al. 2008] Mecca G., Pappalardo A. and Raunich S. - Soluzioni Infrastrutturali Open Source per il Sistema Pubblico di Cooperazione Applicativa. In: *Proceedings of the Sixteenth Italian Symposium on Advanced Database Systems, SEBD 2008*; 2008; Mondello (Palermo). p. 156-167.

[Mecca et al. 2013] Mecca G., Santomauro M. and Santoro D. - Large-Scale Deployment of Middleware-Oriented Government Interoperability Frameworks. In: *Proceedings of the 7th International Conference on Methodologies, Technologies and Tools enabling e-Government (MeTTeG 2013)*; 2013; Vigo, Spain.

[Mecca et al. 2014] Mecca G., Santomauro M., Santoro D. and Veltri E. - On Federated Single Sign-On Systems within e-Government Architectures. Technical Report, Department of Mathematics, Computer Science and Economics; 2014.
http://freesbee.unibas.it/articles/TR2014.pdf.

[Rebollo et al. 2012] Rebollo O., Mellado D. and Fernández-Medina E. - A Systematic Review of Information Security Governance Frameworks in the Cloud Computing Environment. Journal of Universal Computer Science. 2012;18(6):798-815.

[Sabucedo and Anido Rifòn 2010] Sabucedo L.A. and Anido Rifòn L. - Locating and Crawling eGovernment Services A Lightweight Semantic Approach. Journal of Universal Computer Science. 2010;16(8):1117-1137.

[Taylor et al. 2009] Taylor H., Yochem A., Phillips L. and Martinez F. - Event-Driven Architecture: How SOA Enables the Real-Time Enterprise. Addison Wesley; 2009.

[The Java Community Process] The Java Messaging Service (JMS) Specification v. 1.1. [Internet]. http://download.oracle.com/otndocs/jcp/7195-jms-1.1-fr-spec-oth-JSpec/.

[The UK e-Government Unit] UK eGovernment Interoperability Framework (e-GIF). [Internet]. http://www.govtalk.gov.uk/.

[Woolf 2007] ESB-oriented architecture: The wrong approach to adopting SOA. [Internet]. 2007 http://www.ibm.com/developerworks/webservices/library/ws-soa-esbarch.