LARGE-SCALE DEPLOYMENT OF MIDDLEWARE-ORIENTED GOVERNMENT INTEROPERABILITY FRAMEWORKS

Giansalvatore Mecca Michele Santomauro Donatello Santoro
Università della Basilicata
Viale dell'Ateneo Lucano, 10 – 85100 Potenza, Italy
giansalvatore.mecca@gmail.com michelesantomauro@gmail.com donatello.santoro@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 in terms of their internal architectures, which we call "container-bound" and "container-independent". 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 that show how container-bound solutions guarantee better scalability in single-instance deployments, while container-independent solutions represent a better solution in multiple-instance deployments.

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] [2] [3] 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. Most of these frameworks share a number of common features, among which the adoption of Web Services as the underlying technology is the most prominent. In fact, most GIFs rely on *Service-Oriented Architectures (SOA)* [4] as a platform for remote communication and message exchange. However, they often differ quite significantly from one another in scope and philosophy. Broadly speaking, we may classify SOA-oriented GIFs in two main categories.

On one side we have *lightweight* GIFs; these frameworks are oriented towards 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 [1].

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*; in a middleware-oriented framework, systems do not interact directly with each other, 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. Two examples of heavyweight GIFs are the Italian SPCoop infrastructure [5], and the European Interoperability Standard IDA eLink [6].

This paper is focused on GIFs of the second kind, and more specifically on implementations of the Italian SPCoop standard. After quickly reviewing the SPCoop architecture, we concentrate on a crucial feature of middleware-oriented solutions, namely the need for large-scale deployment of instances of the middleware services, also called *domain gateways* [5]. In this respect, we compare two open source implementation of the SPCoop standard. The first one, called freESBee [7] [8], has been developed by our group at University of Basilicata. The second one is called OpenSPCoop [9] [10]. We show how these two systems are based on completely different philosophies in terms of internal design choices, which we call *container-bound architecture* and *container-independent architecture*.

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 architecture of the chosen middleware solution deeply impacts deployment policies. In fact, we discuss two different deployment solutions. The first uses a single middleware instance per server; in this case, container-bound solutions guarantee better scalability due to their strong integration with the services offered by the J2EE container. The second solution deploys multiple middleware instances per server; for this case, container-independent architectures represent a better compromise between scalability and deployment effectiveness. The following sections are devoted to the development of these ideas.

2. – An Overview of SPCoop

SPCoop[1] 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 [11], and a middleware architecture that has the following goals: (a) to standardize the format of messages, and to support the automatic enrichment of message with standard-compliant metadata; (b) to provide automatic routing, and to abstract the addressing protocol with respect to the actual location of services over the network; (c) 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* [12]. However, differently from ESBs, that are typically centralized middleware components for the various information systems of a 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

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

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 belonging to different administrations (either local or central), and therefore to different domains and networks, may exchange messages through the respective domain gateways. In essence, a 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.

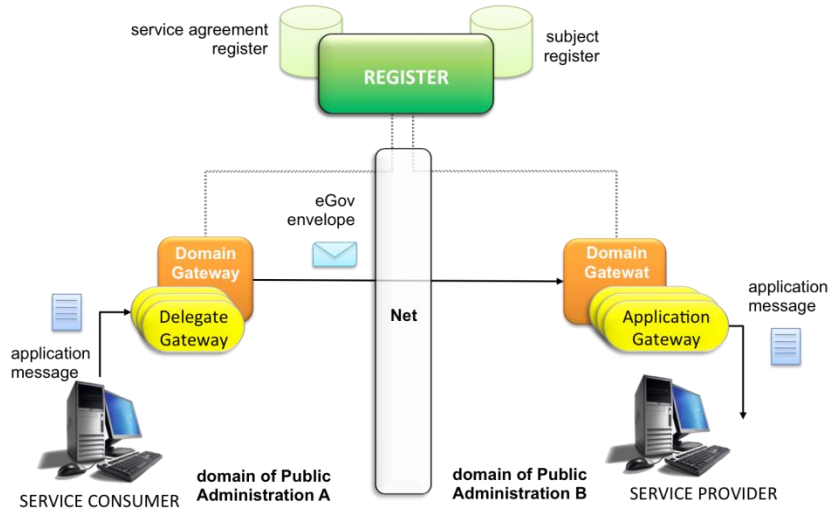

Figure 1: The SPCoop Architecture

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.

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, in order 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: (a) all subjects that are authorized requesters or providers for a given service; (b) all service interfaces (in essence, fragments of WSDL code); (c) a specification of the cooperation profile for each service, either synchronous, asynchronous or one-way; (d) security policies for the service; (e) service level agreements; (f) additional semantic annotations to further enrich the description of the service.

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 [10] has been historically the first open-source implementation of the SPCoop domain gateway. It offers a suite of modules: beside an implementation of the domain gateway, it also provides a registry service, and an event manager to support publish and subscribe applications.

OpenSPCoop is a J2EE Web application developed for the JBoss application server[2]. To understand the relationship between OpenSPCoop and JBoss it is useful to recall that domain gateways are message-oriented applications, i.e., they are conceived to be listening on appropriate network endpoints for incoming messages, receive the messages, process them, and then forward the result to the appropriate 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*) [13] 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, that 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 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,

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

given the strong integration with the J2EE container and the excellent JMS implementation provided as part of JBoss.

4. – freESBee

freESBee [8] 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 behavior of the gateway. Second, while the technical specification is rather prescriptive with respect to the core components, there are other components of the middleware, and more specifically the federated identity management system, the service-level agreement 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;
- 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;
- 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, that can be used to implement event-driven applications based on publish and subscribe.

In addition to this, freESBee makes two main advancements with respect to other implementations, as discussed in the following paragraphs.

4.1. – 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*.

Enterprise Integration Patterns (*EIPs*) [14] 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. Some of the main patterns are shown in Figure 2, along with their graphical notation. Names are mostly indicative of their function, for a detailed description please see www.enterpriseintegrationpatterns.com.

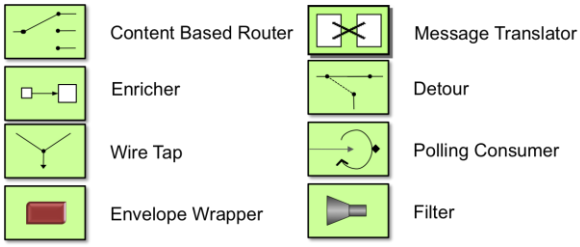

Figure 2: Enterprise Integration Patterns

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.

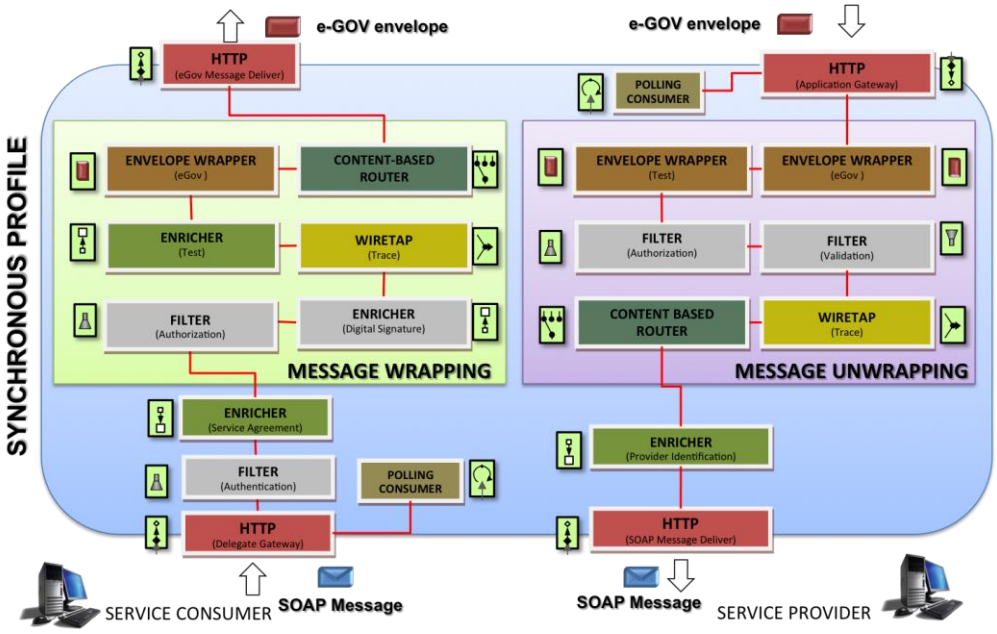

Figure 3: SPCoop Gateway in terms of EIPs

There are in fact, various open-source libraries that support the development of EIP-based applications. In essence, these libraries provide ready-made building blocks that an 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[3], a well-known implementation of EIPs.

4.2. – 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

[3] http://camel.apache.org/

own set of endpoints for delegate and application gateways, based on a Jetty[4] internal engine. This has two main consequences: (a) each freESBee instance is fully independent from the container, since it does not rely on any of its services; (b) 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.

5. – Massive Deployment of Domain Gateways

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.

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.

To solve this problem, 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. This, in turn, generated a significant technical problem: IT departments of Italian Regions were requested to host within their server farms hundreds of different instances of SPCoop domain gateways.

There are three main solutions to host multiple instances of a middleware module within a single server-farm. The first one consists in having a dedicate server for each domain gateway instance; in fact, 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[5], this solution becomes clearly unfeasible; in fact, we shall not considered this any further in the following of the paper.

To reduce the number of servers to deploy, several domain gateway instances need to be deployed within a single server. There are two main strategies to do this:

– 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 dif-

[4] Jetty is a well-known open-source http server available at http://www.eclipse.org/jetty/.
[5] In Italy there are several Regions of this size. To give an example, Region Lombardia includes 1.544 municipalities.

ferent instances would collide with each other when using the messaging service infrastructure offered by the container (messaging queues and endpoints);

– 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 compare these two solutions in the next section.

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 a typical interoperability scenario, with one service provider, one domain gateway, and a number of service requesters. 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.

Service requesters were deployed on a separate physical machine connected to the server through a high-speed local network. 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.

Both the server and the client are dual-core machines with an Intel Xeon Processor at 3GHz, and 8GB RAM. The operating system was Lubuntu 12.10. 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.

We tested the most recent 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.

We also tested the latest available version of freESBee (v. 2.1), deployed according to the standard set of configuration parameters suggested on http://freesbee.unibas.it. We selected Tomcat v. 7.0.35 as a container for freESBee, due to its wide adoption and ease of configura-

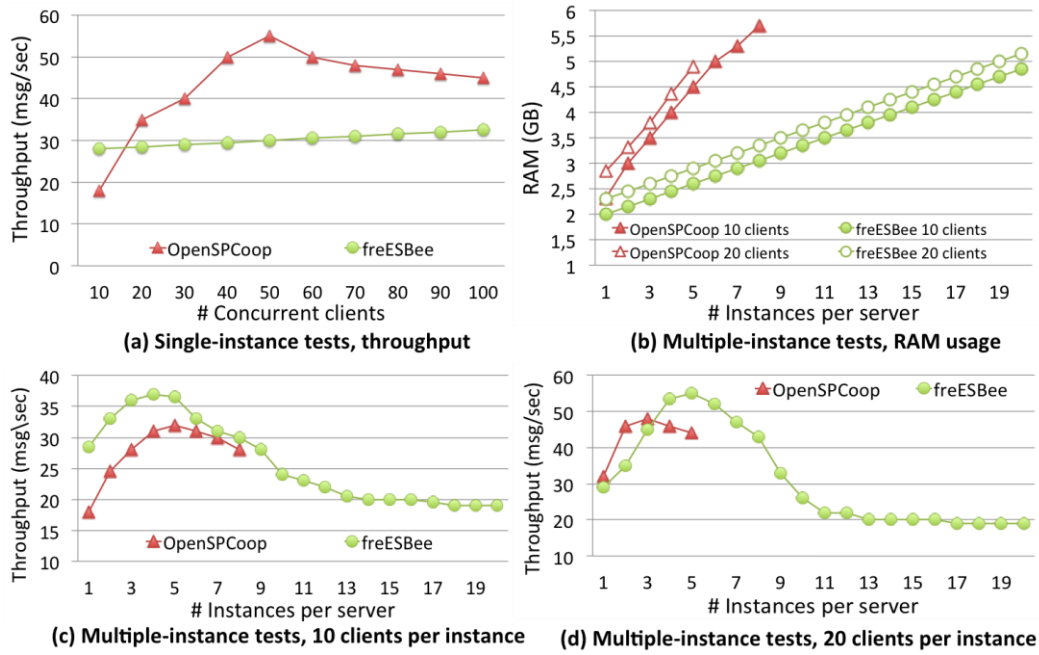

Figure 4: Experimental Results

tion. Multiple-instance tests were run by deploying different copies of freESBee inside the same Tomcat instance.

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 Linux 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.

Figure 4.a reports scalability results for single-instance tests. 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 scales nicely with the number of clients, with a maximum on our server configuration for approximately 50 concurrent clients. Beyond that workload, throughput decreases because the server CPU and the PostgreSQL connection pool become the bottleneck. freESBee had a constant throughput of approximately 30 messages per second, independently of the number of clients. Overall, the tighter container integration of OpenSPCoop guaranteed better performance, since the thread management policy of JBoss is more aggressive than the one used by Camel.

On the contrary, freESBee had better performance in multi-instance tests. Figure 4.b reports RAM occupation for multiple-instance tests ranging from 2 to 20 domain gateway instances per server. Tests were run both for 10 and 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 10-client configuration, the maximum number of OpenSPCoop instances for which the test succeeded was 8; in the 20-client configuration it was 5. Above those numbers some of the instances stall, and high num-

bers 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 4.c and Figure 4.d 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.

Notice that the server configuration we chose is relatively low-end. We believe this is quite close to what typically happens in the server farms of most public administrations in Italy. This is also why we tested up to 20 instances per server. Notice, however, that by adopting high-end configurations for the server this number can be made significantly higher.

To summarize, our experiments show that container-bound solutions may achieve better performance in stand-alone installations. On the contrary, when the number of instances per server increases container-independent solutions provide a better compromise between scalability and deployment effectiveness. We believe these results may be of help to middleware developers and system managers in order to choose the best solution for their interoperability tasks.

References

[1] L. Guijarro, «Interoperability Framewors and Enterprise Architectures in e-Government Initiatives in Europe and the Unites States,» *Government Information Quarterly,* vol. 24, pp. 89-101, 2007.

[2] W. Abramowicz, A. Bassara, M. Wisniewski e P. Zebrowski, «Interoperability Governance for e-Government,» in *Information Systems and e-Business Technologies*, Berlin Heidelberg, Springer-Verlag, 2008, pp. 14 - 24.

[3] N. M. Ibrahim e M. Hassan, «A survey on different interoperability frameworks of SOA systems towards seamless interoperability,» in *International Symposium on Information Technology (ITSim 2010)*, 2010.

[4] D. Krafzig, K. Banke e D. Slama, Enterprise SOA: Service-Oriented Architecture Best Practices, Prentice Hall, 2005.

[5] R. Baldoni, S. Fuligni, M. Mecella e F. Tortorelli, «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*, Tokyo, Japan, 2008.

[6] European Dynamics, «IDA eLink Specification,» November 2004. [Online]. Available: http://ec.europa.eu/idabc/servlets/Doc1a78.pdf?id=18685.

[7] G. Mecca, A. Pappalardo e S. Raunich, «Soluzioni Infrastrutturali Open Source per il Sistema Pubblico di Cooperazione Applicativa,» in *Proceedings of the Sixteenth Italian Symposium on Advanced Database Systems, SEBD 2008*, Mondello (Palermo), 2008.

[8] «The freESBee Project Web Site,» [Online]. Available: http://freesbee.unibas.it/.

[9] A. Corradini e T. Flagella, «OpenSPCoop: un Progetto Open Source per la Coperazione Applicativa nella Pubblica Amministrazione,» in *Atti del Convegno Italiano AICA*, 2007.

[10] «The OpenSPCoop Project Web Site,» [Online]. Available: http://www.openspcoop.org.

[11] DigitPA, «Servizi di Interoperabilità Evoluta,» [Online]. Available: http://www.digitpa.gov.it/spc/servizi-interoperabilit-evoluta.

[12] B. Woolf, «ESB-oriented architecture: The wrong approach to adopting SOA,» IBM Developerworks Technical Library, September 2007. [Online]. Available: http://www.ibm.com/developerworks/webservices/library/ws-soa-esbarch.

[13] Wikipedia, «Java Messaging Service (JMS),» [Online]. Available: http://en.wikipedia.org/wiki/Java_Message_Service.

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