Government Interoperability Frameworks: Middleware Tools for the Italian SPCoop

Donatello Santoro , Michele Santomauro, and Enzo Veltri

Università della Basilicata – Potenza, Italy

(Discussion Paper)

Abstract. We discuss deployment solutions for e-Government Interoperability Frameworks (GIFs). We concentrate on the italian SPCoop, a prominent example of a middleware-oriented architectures, where middleware modules act as intermediaries among information systems. We review the SPCoop architecture, and two popular open-source implementations, called OpenSPCoop and freESBee. The two solutions follow radically different philosophies in terms of their internal architectures, which we call "container-bound" and "container-independent". We compare the two systems with respect to one of the main problems in large-scale deployment of SPCooop-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.

1 Introduction

The development of an electronic Public Administration is a priority for the majority of governments in the world. This is related to the goal of promoting a new paradigm of society known as the *information society*, with the aim of making the PA more efficient, effective and citizen-centred [13]. This goal imposes a fundamental and non-trivial technical requirement to PAs, namely the ability to work in a cooperative way, and therefore 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) [8], where a *service consumer* requires data or applications (also called services) offered by a *service provider*, with an exchange of messages through Web Services organized 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 organizations or Governments that need to coordinate all agencies within one country. These problems are usually solved adopting *Government Interoperability Frameworks* (GIFs). Some of these frameworks 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 standardization, transparent routing, enhanced security, reliability, and quality-of-service monitoring [11]. Many countries of the world have adopted their own GIFs. In general, we may classify SOA-oriented GIFs in two main categories:

(i) *lightweight* GIFs: frameworks based on a point-to-point architecture, in which a service consumer and a service provider directly exchange messages with one another through Web services organized according to a common format. Lightweight GIFs represent the majority in Europe [5].

(ii) *heavyweight* GIFs frameworks: they 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(message-format standardization, transparent routing, enhanced security, reliability, and quality-of-service monitoring). Examples of heavyweight GIFs are the Italian SPCoop infrastructure [2], and the European Interoperability Standard IDA eLink [4].

In this paper we discuss heavyweight GIFs, and more specifically the implementations of the Italian SPCoop standard. After quickly reviewing the SPCoop architecture, we concentrate on the large-scale deployment of the middleware services, also called *domain gateways*. In this respect, we compare two open source implementations of the SPCoop standard: freESBee [10] and OpenSPCoop [3]. We show how these systems are based on different philosophies in terms of internal design choices, which we call *container-bound architecture* and *container-independent architecture*.

One of the main technical challenges to reach a wide-spread availability of middleware-oriented GIFs is 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 and the second solution deploys multiple middleware instances per server.

2 An Overview of SPCoop

SPCoop is the Italian e-government interoperability infrastructure, aimed at enabling the seamless exchange of application services among information systems of Public Administrations. It is based on a set of standards and guidelines [1], and a middleware architecture that has the following goals: (a) to standardize the format of messages; (b) to provide automatic routing; (c) to provide advanced services, like identity management and single sign-on, service level agreements, and publish and subscribe. In this respect, we can say that the SPCoop standard shares a number of similarities with the goals of *Enterprise Service Buses* [14]. However, differently from ESBs, that are typically centralized middleware components for the various information systems, 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*.

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. For example, 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, 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. 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, 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

In this section we discuss two certified implementations of the SPCoop domain gateway: OpenSPCoop and freESBee.

3.1 OpenSPCoop

OpenSPCoop (http://www.openspcoop.org) has been historically the first open-source implementation of the domain gateway. Besides the domain gateway it offers a registry

service and an event manager to support publish and subscribe applications. It is a J2EE Web application developed for the JBoss application server.

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) specification [6] 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, given the strong integration with the J2EE container and the excellent JMS implementation provided as part of JBoss.

3.2 freESBee

freESBee (http://www.freesbee.unibas.it) is an alternative implementation of the SPCoop standard, that was developed at University of Basilicata. The development 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, and it is composed by the following sub-project: (a) freESBee itself is the domain gateway, and offers support for registry services; (b) freESBee-SP[12] 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; (c) 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; (d) 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.

Enterprise Integration Patterns. freESBee using a well-known conceptual model for messaging services, namely *Enterprise Integration Patterns (EIPs)*[7]. These 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. 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. In essence, these libraries provide ready-made building blocks that an application developer can easily customize and compose. freESBee is implemented on top of Apache Camel (http://camel.apache.org), a well-known implementation 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 installation starts its own set of endpoints for delegate and application gateways, based on a Jetty 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) or can also be deployed as a stand-alone application outside of any J2EE container.

4 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

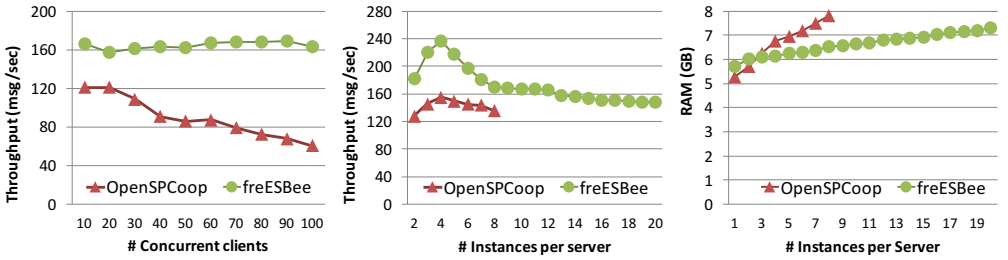
intermediaries for smaller municipalities within their territories. This, in turn, generated a significant technical problem: IT departments of Italian Regions were requested to host within their server farms hundreds of different instances of SPCoop domain gateways. There are three main solutions to 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, this solution becomes clearly unfeasible. 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: (i) *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; 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); (ii) *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.

5 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: (i) first, we measured the throughput achieved by the domain gateway, i.e., the average number of messages per second handled during a test section; (ii) second, we investigated how the throughput varies 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. 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. To simulate the traffic originated by an increasing number of clients, we used Apache JMeter v. 2.7, 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 2.6 GHz, a 512 GB SSD, 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



(a) Single Instance (b) Multiple Instance (c) Multiple Instance RAM usage

Fig. 1. Experimental Results

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 OpenSPCoop (v. 1.4.1), configured according to the standard set of configuration parameters suggested on their website under JBoss. As suggested, 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 version of OpenSPCoop (v. 2.0) under Tomcat.

We also tested the latest available version of freESBee (v. 2.2.6), deployed according to the standard set of configuration parameters. We selected Tomcat 7 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.

Throughput values are the average number of requests completed per second during a test session, computed by JMeter. In addition, we used the operating system 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

Fig. 1(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 v.1.4.1 under JBoss reaches a throughput of 120 messages 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.2.0 under Tomcat to confirm the results, and no noticeable change of performance was recorded. These experiments show also how freESBee outperforms OpenSPCoop, reaching a maximum of approximately 150 messages per second.

Figures 1(b-c) report results for multi-instance tests on the loopback scenario. As it was expected, freESBee had better performance in multi-instance tests. Fig. 1(c) reports RAM occupation for multiple-instance tests ranging from 2 to 20 domain gateway instances per server. Tests were run for 10 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. 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. Fig. 1(b) 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.

References

1. AGID. Servizi di interoperabilità evoluta. http://www.agid.gov.it/agenda-digitale/infrastrutture-architetture/sistema-pubblico-connettivita, 2014.
2. R. Baldoni, S. Fuligni, M. Mecella, and F. Tortorelli. The italian e-government enterprise architecture: a comprehensive introduction with focus on the sla issue. In *Service Availability*, pages 1–12. Springer, 2008.
3. A. Corradini and T. Flagella. Openspcoop: un progetto open source per la cooperazione applicativa nella pubblica amministrazione. *Atti del Convegno Italiano AICA*, 2007.
4. E. Dynamics. Ida elink specification. http://ec.europa.eu/idabc/servlets/Doc1a78.pdf?id=18685.
5. L. Guijarro. Interoperability frameworks and enterprise architectures in e-government initiatives in europe and the united states. *Government Information Quarterly*, 24(1), 2007.
6. M. Hapner, R. Burridge, R. Sharma, J. Fialli, and K. Stout. Java message service. *Sun Microsystems Inc., Santa Clara, CA*, 2002.
7. G. Hohpe and B. Woolf. *Enterprise integration patterns: Designing, building, and deploying messaging solutions*. Addison-Wesley Professional, 2004.
8. D. Krafzig, K. Banke, and D. Slama. *Enterprise SOA: service-oriented architecture best practices*. Prentice Hall Professional, 2005.
9. G. Lodi, A. Maccioni, and F. Tortorelli. Spcdata: the italian public administration data cloud. *Information and Communication Technologies in Public Administration: Innovations from Developed Countries*, 195:255, 2015.
10. G. Mecca, A. Pappalardo, and S. Raunich. Soluzioni infrastrutturali open source per il sistema pubblico di cooperazione applicativa. In *SEBD*, pages 156–167, 2008.
11. G. Mecca, M. Santomauro, D. Santoro, and E. Veltri. Middleware-oriented government interoperability frameworks: A comparison. *Journal of Universal Computer Science*, 20(11):1543–1563, 2014.
12. G. Mecca, M. Santomauro, D. Santoro, and E. Veltri. On federated single sign-on in e-government interoperability frameworks. *To Appear in International Journal of Electronic Governance, Special Issue on MeTTeG 2015*, 2016.
13. D. Soares and L. Amaral. Information systems interoperability in public administration: Identifying the major acting forces through a Delphi study. *Journal of Theoretical and Applied Electronic Commerce Research*, 6:61–94, 2011.
14. B. Woolf. Esb-oriented architecture: The wrong approach to adopting soa. *IBM Developerworks Technical Library, September*, 2007.