Bulletin of the *Transilvania* University of Braşov
Series I: Engineering Sciences • Vol. 5 (54) No. 1 - 2012

EVALUATION OF SOFTWARE SERVICE FRAMEWORKS FOR INDUSTRIAL APPLICATIONS

A. GÎRBEA[1] C. SUCIU[1,2] F. ŞIŞAK[1]

Abstract: *During the last decades software (web) services have drawn the attention of all big corporations mainly due to their flexibility, reusability and adaptability. The paper evaluates three of the most popular service frameworks: Apache CXF (SOAP web services), Jersey (REST web services) and Apache River (or Jini-Java services) using a novel industrial architecture. Two performance tests have been developed and as a result Apache River services are more than one order of magnitude faster than both web service frameworks, which are comparable to each other. Thus Jini Java-based services should be used whenever possible in industrial applications.*

Key words: *service, industry, SOAP, REST, Apache River.*

1. Introduction

Over the last decades, the rate of activity has greatly increased and many industrial companies have to adapt their technologies in order to cope with all these demands. Therefore the most important aspects which have drawn the companies' attention are the adaptability, flexibility and reusability. The software (web) services meet all these needs and they are gradually becoming the key points of the world where devices [11] are interconnected in different ways.

Some of the challenges [5] and demands concerning the device ecosystems which have been identified in the specialized literature are the following: devices should provide universal, interoperable and secure access interfaces; web standards should be used whenever possible; devices should be easy to integrate into complex systems and the integration should be scalable; every subsystem should be exposed as a device capable of being integrated into more complex systems; any device should be reusable at any level; any device should have a high-level management interface, which facilitates configuration, monitoring, fault diagnosis and maintenance etc.

Over time, the global corporations have tried to fulfill all the mentioned demands therefore many projects have been developed and tested. One important project [1] has been SIRENA (Service Infrastructure for Real-time Embedded Networked Devices). Through this project Schneider Electric has produced an early Devices Profile for Web Services (DPWS) implementation targeted at embedded devices.

Filho, et al. [2] have used web services to directly communicate with device controllers (PLCs). They have acknowledged the fact

[1] Dept. of Automatics, *Transilvania* University of Braşov.
[2] Siemens Corporate Technology, Romania.

that OPC (OLE - Object Linking and Embedding - for Process Control) is the main means of allowing access from the higher levels of the automation pyramid to the devices, but since classic OPC was platform and technology dependent, they have replaced the OPC level and introduced web services developed using the Java Native Interface and Apache Axis.

The European Union funded SOCRADES [10] (Service-Oriented Cross-Layer Infrastructure for Distributed Smart Embedded Devices) project is based on the principle of collaborative automation. It divides the industrial enterprises in four layers: device, composition, middleware, and an application layer. Then, at the composition layer, several embedded devices may be combined in order to offer value added functionality. This functionality is also offered as web services using DPWS.

The paper intends to offer a better understanding of software services and to evaluate the currently most popular approaches for creating services: SOAP, REST and Apache River (Jini) regarding their applicability in industrial applications. These three types of services have been tested on a novel industrial architecture which was previously introduced in [6], and whose goal is to improve manufacturing in factories through the determination of an optimized production schedule and then by using the solutions of this plan to automatically manufacture the requested products.

The paper is organized as follows. The second section offers a brief description regarding SOAP and REST web services approaches and it also describes the Jini Java services. In the third section, the architecture proposed for the evaluation of the service frameworks is introduced. The fourth section presents the performance tests and the obtained results and finally the last section draws the conclusions.

2. Software Service Frameworks

Choosing a service framework is always a difficult task. Several open source frameworks are available, some of the most widely used ones being: Apache CXF, Jersey, Apache River (Jini), Apache Axis, GlassFish, Metro, JBossWS etc.

Each service framework aims to provide a robust infrastructure so that the developer can build, deploy and publish services as simple as possible. Next, three of the most important service frameworks, which will be evaluated through the architecture described in section 3, will be introduced: Apache CXF (SOAP based WS), Jersey (REST based WS) and Jini (java based services).

The concept of (web) services implies three types of roles - a service consumer, a service provider and a service registry (optional component). Figure 1 displays the interactions of three roles.

The service providers supply services and they respond to the requests of the service consumer. The service consumer uses the services offered by the service provider.

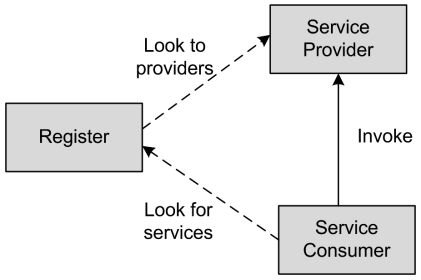

Fig. 1. *The roles and types of interactions inside service frameworks*

In 1998 Microsoft has developed an object access protocol called “Simple Object Access Protocol” (SOAP) which was designed to be a platform and language-neutral alternative to previous middleware technologies like CORBA and

DCOM. The SOAP specification is currently maintained by the XML Protocol Working Group of the World Wide Web Consortium. For SOAP - based web services the service provider is responsible for publishing the service contract (WSDL file) on the web, from where the consumer can access it directly or by searching for it in the web services registry. Usually the service consumer generates the code of the web service client starting from the WSDL file and through the use of the tools offered by web services frameworks [7].

REST (Representational State Transfer) is a style of software architecture for distributed hypermedia systems such as World Wide Web and it was first introduced in 2000 by Roy Fielding in his doctoral dissertation. The REST architectural style was developed in parallel with HTTP/1.1, based on the existing design of HTTP/1.0. For REST - based web services there is no formal contract between the provider and the consumer of the web services. In this case the consumer has to know the message format (for example XML) and the operations supported by the provider. The service provider exposes its set of operations through the use of HTTP methods: GET and POST. The service consumer will then invoke one of these methods using the URI (Uniform Resource Identifier) and the HTTP protocol [9].

The Jini (also called Apache River) network architecture was introduced by Sun in 1998 and its goal was to construct distributed systems in the form of modular co-operating Java services. Jini is a service-oriented architecture, it is not a acronym and does not have a special meaning. A programming model, which extends the Java technology, is defined inside the Jini technology, thus allowing the creation of distributed, security-enhanced systems composed of both services and clients [8].

2.1. SOAP-based APACHE CXF web services

CXF has been designed by “Apache Software Foundation” and resulted from the merger of two open source projects: Celtix developed of Iona Technologies and XFire developed by a team hosted at Codehaus ("CXF" comes from combining "Celtix" and "XFire"). Apache CXF is open source and enables the development of services using different APIs, like JAX-RS or JAX-WS. The services may be based on different protocols, like SOAP, XML/HTTP, RESTful HTTP, or CORBA and may use different transport protocols, like HTTP, JMS or JBI. This framework has been used in order to create SOAP based web services.

An important feature of CXF is that it offers tools to facilitate the conversation between JavaBeans, web services and WSDL. These tools help developers to generate web service clients based on Java and JavaScript starting from WSDL or to generate a WSDL file starting from a service implementation. CXF also supports Maven and Ant integration.

CXF is easy to use because it provides complete support for integration with the Spring framework, so that a POJO (Plain Old Java Object) class exposed as web service by using the CXF framework can fully utilize the benefits of the Spring framework (e.g. transaction capabilities can be applied declaratively to POJO classes representing web services through the Spring infrastructure). Using the Spring framework, the entire configuration of the web services is simplified, and the uploading of the services on the server is simplified through the use of XML configuration files

CXF provides a flexible way to develop and test the services in an independent environment, and then to upload them on a server application. The web services

developed with CXF may be uploaded on servers like Tomcat or J2EE based containers such as WebSphere, WebLogic, JBoss, Geronimo and Jonas. CXF also provides integration with SCA (Service Component Architecture) containers such as Tuscany [7].

2.2. REST-based Jersey web services

Since its appearance in 2000, the REST approach has gradually gained more and more attention. More and more companies like Amazon or Google have switched from the traditionally SOAP based web services to the so called REST services.

These changes happened due to the fact that REST web services are lightweight, easy to use and declarative. They are *lightweight* because the REST approach manages the business data and functionality as identified resources. The service answer is the resource representation and it doesn't need additional encapsulation such as a SOAP envelope. Furthermore the REST architectural style is considered a communication technique which uses only HTTP and XML. Therefore the REST services are directly using the HTTP protocol and they offer a uniform user interface through the use of the same set of methods. These types of services are *easy to use* because there is a standard set of HTTP status codes which are used in order to understand the response of the invocation. Moreover, the REST web services are focusing on their own data and resources, hence they are known as *self-declarative*.

For the development of the REST-based web services, the Jersey framework has been used. It is maintained by an open source community which builds an implementation of JSR-311: JAX-RS (Java API for RESTful Web Services). Thus the RESTful web services are easy to build using the support for the annotations which are defined in JSR-311. Further, the JSR suit can be extended through the use of the additional API provided by Jersey (which is not specified by JSR-311) [9].

2.3. Jini-based Java services

Jini has appeared due to the work put in the development of Java, the main aim being to simplify the distributed computation. Therefore the Jini system is a distributed system based on the idea of federating groups of users and the resources required by those users. The goal is to turn the network into a flexible and easy administrated tool where the resources can be easily found by the clients. *Resources* could be understood either as hardware devices, software programs or even a combination of the two.

We have to emphasize that the Jini technology infrastructure is a Java technology. The Jini architecture gains much of its simplicity from assuming that the Java programming language is the implementation language for components. Many features of the Jini technology focus on the ability to dynamically download and run code. However, the Java technology-centered nature of the Jini architecture is stronger related to the Java application environment than to the Java programming language. The Jini system supports any programming language if a proper compiler is provided which is able to generate the corresponding bytecodes for the Java programming language.

The Jini services are found and resolved by a *lookup service*, which represents the interface between the system and its users. Its role is to map interfaces, which provide the functionality of the service, to instances which implement the service. Through descriptive attributes, which can be associated to a service, the selection of services is simplified and leads to better

choices. The heart of a Jini system is represented by the following protocols: *discovery*, *join* and *lookup*. The pair of protocols - discovery and join - occurs when a device is plugged in. Discovery occurs when the service is looking for a lookup service with which to register. Join occurs when a service has located a lookup service and wishes to join it. Lookup occurs when a client or user needs to locate and invoke a service described by its interface type and possibly other attributes.

In conclusion a Jini system represents a collection of clients and services which are communicating through the Jini protocols. The most used approach applied for the communication between Java applications is based on the Java Remote Method Invocation mechanism [8]. One of its main advantages is that it enables the passing of full objects, including code, from one instance to another (this advantage is fully exploited by the Jini system).

3. A Service-Oriented Architecture for the Optimization of Industrial Applications

In the following an architecture built around the corner poles of SOA (Service Oriented Architecture) is introduced. Figure 2 displays the architecture, which is composed of three main levels, and the interactions between these three levels. The lowest level of the architecture is represented by an OPC UA server (OLE for Process Control Unified Architecture) which models the data from the device level so that every important piece of information becomes easily accessible in a unified way. In [6] a detailed description regarding the OPC Unified Architecture can be found, standard which was introduced as a real replacement for the COM existing specifications without losing the features or performance. The second level of the architecture is represented by the services which are organized in two sublevels.

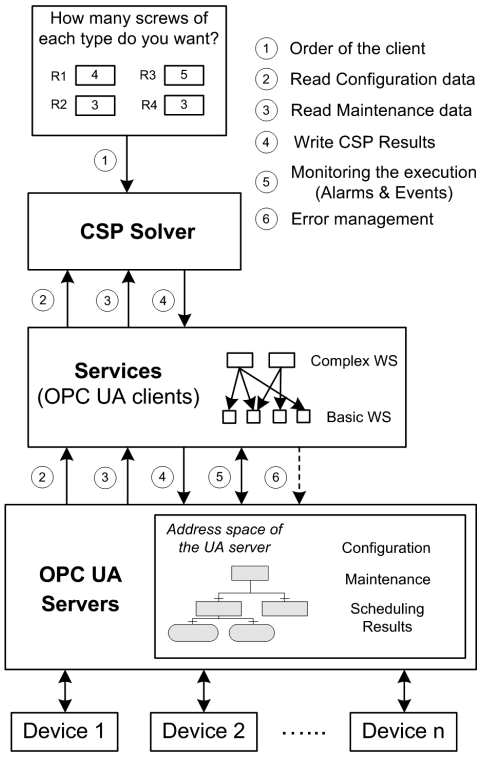

Fig. 2. *Main components of the architecture*

The idea of developing SOA in junction with an OPC UA server seems natural because the OPC UA server has been built around the corner poles of SOA.

The first level is composed of basic services which perform operations like reading, writing etc. and which can be integrated in any scenario (a total of eight basic services have been implemented and provide an additional abstraction of the artifacts lying at lower levels). The second level is composed of complex services. Concrete examples of such complex services may be: “packing”, “dyeing”, “milling” etc. Usually these complex services interact with the basic services to perform the necessary tasks. The goal of the services developed at this level is to allow an easy communication between the highest level of the architecture and the

OPC UA server. Therefore they are used in order to transport the solutions offered by the CSP models, lying at the top level, to the UA server and thus to facilitate the automated manufacturing of the products. Besides that, before determining the optimized solutions, the CSP model calls the services in order to determine the current state of the physical devices and if it is necessary to reconfigure the model (e.g. if maintenance activities are performed for certain stations).

On the other side, the services can be used in order to determine the values of key performance indicators (KPI) for the enterprise level of the company. Due to the introduction of the services at the second level, the whole architecture becomes more flexible and smaller reaction times are assured when there is need for changes.

The third level is represented by the CSP models. The constraint programming paradigm is used in order to solve combinatorial optimization problems. A CSP instance, usually called model, is described by a set of variables, a set of possible values for each variable and a set of constraints between variables. Constraints are used not only to test the validity of solutions but also to reduce the computational effort needed to solve combinatorial problems. This process is called *constraint propagation*. The second component of the CSP solvers is the *heuristic search* with backtracking strategy adopted. Most of the problems modeled through the CSP approach are planning and scheduling problems, whose goals are to allocate scarce resources to different activities. A detailed description of the implementation of the architecture is available in [3].

4. Performance Tests and Results

The architecture described in section 3 has been used to test the performance of the three service frameworks, Apache CXF, Jersey and Apache River/Jini. The services developed through the three different frameworks, were implemented using the Java language, Eclipse IDE and the Tomcat server.

4.1. Single service execution times

The first test evaluates the execution times of the basic services and Table 1 displays the average execution times together with the standard deviations. The functionalities performed by the services specified in Table 1 are described in detail in [4]. The time specified in parentheses represents the execution time of the first call, while the average execution times have been determined only for the subsequent calls. Only the first four services are time critical, while the last four ones are related to historical data and are used in order to determine KPI values. As is displayed in Table 1, there is an important difference between the fist call

Basic service execution times Table 1

Basic service	Apache CXF [ms]	Jersey REST [ms]	Apache River (Jini) [ms]
WriteVariable	280.9 ± 10.5 (1102.8)	309.7 ± 7.59 (907.7)	10.1 ± 0.94 (1097.3)
ReadVariable	244.2 ± 2.83 (1259.3)	233.0 ± 3.36 (1312.1)	6.2 ± 0.41 (1142.8)
CallMethod	451.3 ± 32.7 (1167.6)	468.5 ± 23.9 (1078.3)	13.4 ± 0.56 (1112.5)
ReadAlarmState	245.4 ± 2.56 (1243.7)	235.9 ± 2.78 (1308.2)	6.6 ± 0.32 (1245.9)
ReadAlarmEvent	389.8 ± 15.8 (1197.5)	377.4 ± 12.3 (1296.2)	7.6 ± 0.48 (1215.3)
ReadRawData	425.6 ± 6.84 (1223.8)	418.5 ± 10.6 (1287.4)	8.2 ± 0.71 (1267.0)
ReadProcessed	518.8 ± 13.7 (1217.1)	517.6 ± 12.2 (1278.8)	8.3 ± 0.62 (1208.1)
ReadAtTime	278.3 ± 11.9 (1145.0)	272.2 ± 8.11 (1285.3)	7.9 ± 0.69 (1207.3)

of a service made by a user and all subsequent calls. This difference is due to the fact that during the first call performed by each service client, a connection to the UA server has to be established.

The obtained results demonstrate that Apache River services are considerably faster (more than one order of magnitude) than both web service frameworks, which are comparable to each other. In terms of web services, the write operations are performed faster with CXF web services and read operations are faster with Jersey services.

Regarding the first service call, all three frameworks perform similarly because most of the execution time is given by the time needed to connect to the UA server. On the other hand, REST services are easier to develop and are more lightweight than the other two. There is no need for WSDL files (as for Apache CXF) or development of several supportive classes (as for Jini).

4.2. Reading and writing of Boolean variables

The second service specific test has been performed in order to evaluate the performance of the read and write services for boolean variables which are most often used in industrial applications. All boolean variables are stored in groups of 16 inside *Int16* variables. Thus several boolean variables can be written simultaneously and the time required to write a high number of such variables is significantly reduced.

Table 2 displays the results for the three frameworks. As expected, Apache River services are again much faster than the web service frameworks. Regarding the web services, again the write operations are performed faster with CXF web services and read operations are faster with Jersey services.

Table 2
Reading and writing of Boolean variables

Basic web service	**Apache CXF [ms]**	**Jersey REST [ms]**	**Apache River (Jini) [ms]**
Read 100 Boolean variables	1059 ± 34	918 ± 32	37.8 ± 2.1
Write 100 Boolean ariables	1217 ± 28	1340 ± 57	59.3 ± 3.2

Having performed these two tests the decision has been taken to use Apache River services inside the architecture described in section 3 in order to assure shorter communication and reaction times. Although the execution times when using Apache River services are low, one has to keep in mind that all time-critical operations are performed on the controller devices and the correctness of the sequence of operations does not depend on the response times displayed in the two tables.

5. Conclusions

The goal of this paper has been to determine the most suited service framework (SOAP, REST or Jini) for industrial applications. Therefore two performance tests have been developed, for all these three approaches, and they have been applied for the architecture described in section 3.

Jini Services' object-oriented engineering approach produces software that is modular, structured, well documented, and easy to maintain. Jini services have achieved an exceptional reputation in the industry for developing software applications which are efficient, accurate, and reliable in use. For all these reasons and because the performed tests have proved that they are considerable faster than web services, we have chosen to use Apache River (Jini) framework for our future work.

Regarding the web services there are no big differences between the performances of the Apache CXF (SOAP) and Jersey (REST) web service frameworks. The choice between SOAP and REST depends on application requirements. The REST approach is preferable when the application task is to transmit and receive simple XML messages. The SOAP approach is used when the application task consists of numerous contracts to be defined and negotiated between producers and consumers by using WSDL and when there are requirements regarding the adherence to various WS specifications such as those related to security (which are essential for global corporations).

Acknowledgements

This paper is supported by the Sectoral Operational Programme Human Resources Development (SOP HRD), financed from the European Social Fund and by the Romanian Government under the contract number POSDRU/88/1.5/S/59321.

References

1. Bohn, H., Bobek, A., Golatowski, F.: *SIRENA - Service Infrastructure for Real-time Embedded Networked Devices: A Service Oriented Framework for Different Domains*. In: Proceedings of the International Conference on Networking, International Conference on Systems, and International Conference on Mobile Communications and Learning Technologies, Morne, 23-29 April, 2006, p. 43-47.
2. Filho, F.Sd.L., da Fonseca, A.L.T.B., et al.: *Industrial Processes Supervision Using Service Oriented Architecture*. In: 32nd IEEE Conference Industrial Electronics, Paris, 6-10 November, 2006, p. 4552-56.
3. Gîrbea, A., Suciu, C., Sisak, F.: *An Innovative and Flexible Architecture for Industrial Automation*. In: Proceedings of the 13th International Conference on Optimization of Electrical and Electronic Equipment, Braşov, 24-26 May, 2012, in press.
4. Girbea, A., Suciu, C., Sisak, F.: *Remote Monitoring and Control of a Flexible Manufacturing System through a Service Oriented Architecture*. In: Proceedings of the 10th IEEE RoEduNet Conference, Iaşi, 23-25 June, 2011, p. 1-6.
5. Jammes, F., Smit, H.: *Service Oriented Paradigms in Industrial Automation*. In: IEEE Transaction on Industrial Informatics **1** (2005) No. 1, p. 62-70.
6. Mahnke, W., Leitner, S.H., Damm, M.: *OPC Unified Architecture*. Berlin. Springer Press, 2009.
7. Apache CXF. Available at: http://cxf.apache.org/. Accessed: 20.10.2011.
8. Apache River SDK. Available at: http://river.apache.org/. Accessed: 06.11.2011.
9. Jersey REST SDK. Available at: http://jersey.java.net/. Accessed: 16.11.2011.
10. Service-Oriented Cross-Layer Infrastructure for Distributed Smart Embedded Devices (SOCRADES). Available at: http://www.socrades.eu/. Accessed: 20.04.2011.
11. Web services for devices. Available at: http://www.ws4d.org. Accessed: 25.04.2011.