Performance Evaluation of the DORII Instrument Element Data Transfer Capabilities

Luca Berruti, Franco Davoli, Stefano Vignola, and Sandro Zappatore

Abstract The possibility of accessing and managing remote laboratories by exploiting the facilities offered by the Internet represents a challenging issue. To this purpose, many research projects have been funded and different hardware/software solutions have been proposed. A possible approach to the problem may consist of adopting a grid-based platform to expose and share the resources present in a group of laboratories in a uniform way. Recent projects have proposed a complex and complete architecture that enables Internet users to remotely control instrumentation and perform experiments. According to such architecture, the instrument element (IE) represents the basic abstraction that allows physical instrumentation to be exposed as a web service and to become a manageable resource in a grid environment. This chapter aims at evaluating the performance of the transfer of data generated by the instruments from the IE to a multiplicity of users. The evaluation is performed under different traffic load values, generated by the acquisition of measurements, with different numbers of clients connected to the system, and by means of two different data transfer methodologies supported by the IE. The results related to a large number of tests are presented and discussed.

1 Introduction

Remotely controlling a laboratory and the related instruments and devices, sending commands, and acquiring measurements are not new activities in the instrumentation and measurement scenario – they have been performed in a whole range of different applications. However, the goals of recent remote instrumentation service

L. Berruti • F. Davoli (✉) • S. Vignola • S. Zappatore
DIST-University of Genoa/CNIT, University of Genoa Research Unit, Genoa, Italy
e-mail: luca.berruti@cnit.it; franco@dist.unige.it; stefano.vignola@cnit.it; sandro.zappatore@unige.it

paradigms, such as those proposed by GRIDCC [1], RINGrid [2], and DORII [3], three recent projects funded by the European Community, are more ambitious. Indeed, they aim at:

- Providing a set of standard capabilities to perform whatever functionality may be required
- Constructing suitable abstractions of the remote instrumentation, in order to make it visible as a manageable resource
- Presenting the user standard interfaces, which allow browsing the "distributed laboratory space," choose different pieces of equipment, configure their interconnection, orchestrate experiment executions, and collect, process and analyze the results – all by providing also built-in security services

In order to accomplish such tasks to a full extent, laboratories, devices, and instruments should be exposed as a set of web services, whose interfaces may be compliant with some grid architecture, much in the same way as computing and storage devices are. In this way, they would take advantage of: (a) isolation from and relative independence of the underlying networking infrastructure providing connectivity, (b) tools for resource allocation and management, (c) standard user interfaces, and (d) nontrivial quality of service (QoS) control. All the previously mentioned concerns have been the subject of recent interest, strictly connected to the issue of remote instrumentation services (RIS). The widespread diffusion of such services can foster the use of sophisticated and costly scientific equipment, and of *eScience* applications. Furthermore, it should be highlighted that this paradigm does not only apply to large-scale laboratories and devices, but it can be fruitfully employed even with smaller and relatively widespread measurement instrumentation, as often adopted in engineering applications. The interest in this field is witnessed by a number of papers (see, e.g., [4–13]) available in the literature and dealing with these topics.

A possible approach for controlling and managing remote laboratories and instrumentation may consist of employing the grid-based architecture initially developed with the GRIDCC project. According to this architecture, the main role is played by a software component called the instrument element (IE). It represents the basic abstraction that allows physical instrumentation to be exposed as a web service and to become a manageable resource in a grid environment. Through the presence of one or more instrument managers (IMs), the IE actually interfaces the instrumentation, both hiding the details of the drivers from users, and possibly embedding popular proprietary solutions (e.g., LabView), to provide a unified standard interface.

After initial implementations, the IE has been redesigned within the DORII project to simplify and better organize its structure and enhance its performance. A similar enhancement has been undertaken for another basic component of remote instrumentation services, namely, the VCR (virtual control room), a web portal that provides user access to the distributed environment.

The present chapter aims at evaluating the performance of the transfer of data generated by the instruments from the IE to a multiplicity of users, under two different data transfer methodologies supported by the IE. To this purpose, we emulate the generation of measurement data from an instrument as an array of variables, under different load values and with different numbers of subscribing clients. Java message service (JMS) is the publish/subscribe mechanism adopted, and the generation of data is effected by using JMeter, a Java application designed to measure performance.

There are basically two ways in which the combination of these architectural elements can be used to interact with the physical instruments. These are: (a) configuring and setting parameters and (b) collecting measurement data from the instruments. With respect to the second point, possible choices are polling the instrument from the client, by traversing the whole web service protocol stack (which might hinder performance) or using a publish/subscribe approach, whereby clients which subscribed for a certain topic are notified asynchronously by means of a messaging system.

This chapter is structured as follows. Section 2 illustrates the overall DORII architecture, pointing out the role played by its principal components. Section 3 describes the experimental setup employed to carry out the measurement campaigns devoted to evaluate the performance of the data collection from the IE. The results obtained are presented and discussed in Sect. 4. Finally, in the last section conclusions are drawn.

2 Overall DORII Architecture

The overall DORII architecture is diagrammatically depicted in Fig. 1. The architecture stems from the grid-based platform adopted within the GRIDCC project and represents the natural evolution of the latter. Though the GRIDCC platform may include components (called "elements") of different kinds, in the context of distributed remotely controlled laboratories, the main rules are played by the instrument elements (IEs) and the virtual control room (VCR). The latter constitutes the actual interface exploited by the user to carry out any sort of actions and/or measurements that the laboratory, handled by the IE, is able to perform. The IE consists of a web service module embedded in a grid middleware, specifically gLite. The IE actually controls all the devices/instruments present in the laboratory through a set of instrument managers (IMs).

The IE runs in the Apache Tomcat container as an Axis web service and it includes two ancillary sub-components (the access control manager – ACM – and the information and monitor service – IMS), plus a set of IMs that actually handle the instrumentation. The grid framework also provides another data exchange protocol (GridFTP), which allows IMs to send their outputs (in a batch fashion) to other grid elements (e.g., storage elements, SEs). The main task of an IM is to drive and handle a specific class of devices or instruments: obviously, each class of devices requires a specific IM implementation. Under this perspective, an oscilloscope, independent of its manufacturer, will require an "oscilloscope IM," as well as a spectrum analyzer

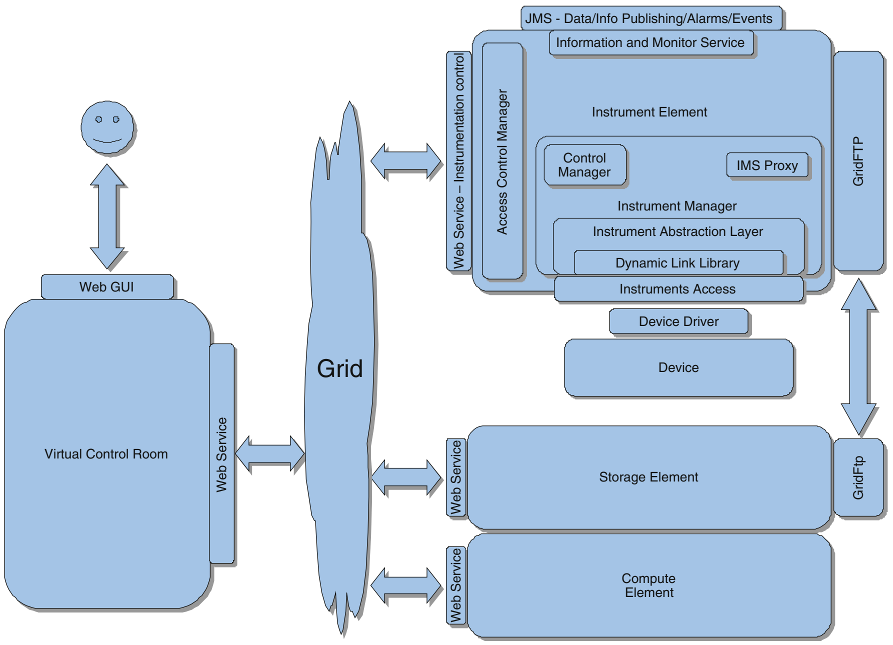

Fig. 1 Overall DORII architecture

will need a "spectrum analyzer IM." In order to handle the instrumentation of a laboratory, the IM exploits a set of application program interfaces (viz., APIs) implemented as a software library, called instrument abstraction layer (IAL). The adoption of the IAL permits us to develop an IM without considering the specific technology used to interface the real device/instrument; in other words, an IM programmer does not need to take care of the hardware specifications at the physical layer, the low level communication protocol, and so on.

As regards the command and data exchange, the requesters (viz., users at the VCR end) can send SOAP requests and get SOAP responses from the IE over a HTTP connection. Upon receiving a message (i.e., a request), the IE dispatches it toward the IM referenced in the message. According to this mechanism, a possible way a user can read the current value of a certain quantity (namely, the "attribute" of a "variable") acquired by a device/instrument consists of triggering a read operation (specifically, a getAttribute method) by means of a user request: hence, a near real-time monitoring may involve a continuous polling of the IE and, in turn, of the appropriate IM.

Employing the getAttribute method is surely more suitable for asynchronous (nonpersistent) reading of an attribute, since a continuous polling may produce an overburden of the system hosting the IE/IM and an overall wastage of resources.

If a continuous (though still asynchronous) monitoring is needed, the IE may take advantage of the use of a publish/subscribe mechanism, as the one provided by a JMS server (sometimes also called a JMS broker) in order to fast deliver data messages from the instrument managers to the users or to the VCR. According to the JMS terminology [14], the messaging system provided by a JMS server may use a managed object, called a "Topic," to identify a specific message flow. A message producer acquires a reference to a JMS topic on the server, and sends messages to that Topic. When a message arrives, the JMS provider is responsible for notifying the presence of a new message to all consumers who have subscribed to that Topic.

According to this scheme, if an attribute is declared "subscribable," the IM internally polls the variable associated to the attribute and, only upon changes of the variable, the IM publishes the new value by means of the services offered by a JMS server "coupled" with the IE.

3 Experimental Setup

The experimental setup adopted to evaluate the performance of the data collection from the IE is sketched in Fig. 2. The PC denoted as "Grid-Node" hosts the IE (ie-release2.1), its related IMs, and the JMS broker (Sun JMS Message Queue 4.3). The possible actions a user can perform, both directly and through a VCR, are mimicked by a JMeter (jakarta-jmeter-2.3.4) application [15] hosted in the PC denoted as "Client Station." More precisely, the grid-node is a Fujitsu-Siemens workstation "Celsius 460," running a Windows-XP sp3 operating system, and the Client Station is a Dell XPS 1,330 laptop running a Debian GNU/Linux 5.0 operating system. The two PCs are directly linked via an Ethernet interface at 100 Mbit/s.

It should be highlighted that in order to effectively estimate the performance of data gathering, no real device is connected to the IMs, thus avoiding possible

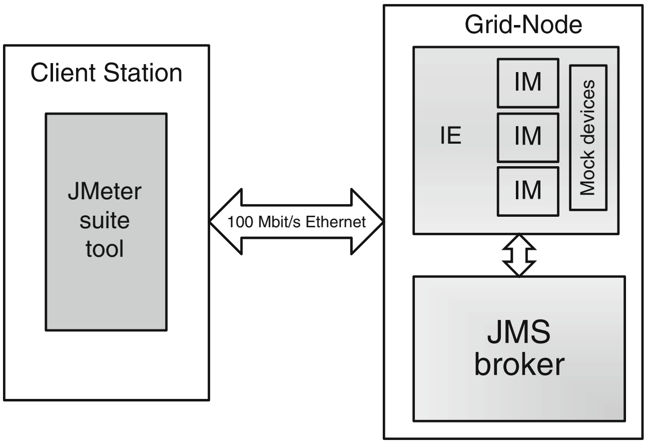

Fig. 2 Experimental setup used during the performance measuring tests

bottlenecks ascribable to the protocols and/or technologies employed to connect the IMs to the devices. Therefore, any IM sends commands and receives responses and data to/from a “mock” device that, upon receiving a data request from the IM, simply assembles a vector of doubles (8 bytes in a 32 bit x86 Intel architecture) of predefined length. A specific application of the IE/IM abstractions to physical instrumentation can be found in [12, 13].

JMeter is a pure Java application originally written to test functionalities of static and dynamic resources, such as Java objects, data base queries, Servlets, and successively expanded to evaluate the performance of a wide class of objects, including SOAP/web services. Nowadays, JMeter represents a quasistandard, well-known tool for performance evaluation able to run in a variety of environments, to provide fast operation and precise timings, and to allow great personalization.

4 Performance Evaluation

The performance of the data collection from the IE has been evaluated in two different operating contexts. According to the former operative mode (hereafter referred as “Polling Mode”), the JMeter mimics a certain number of users that continuously pool the IE, which transmits back a vector of doubles. The term “continuously” means “at the maximum speed supported by the system”; viz., upon receiving a response from the IE, the client promptly sends a new request to the IE. Obviously, the maximal frequency of user requests will depend on several factors, such as: the number of clients which are issuing the requests, the length of each answer, the computation power of the computer hosting the IE, and so on. The purpose is to estimate the average response time (viz., the time spent by a user to receive a vector of data from the IE) versus the number of users and the length of the vector of doubles, and to evaluate the overall traffic offered to the network.

In the latter operative context (hereafter referred as “JMS Subscribing Mode”), the JMS broker is exploited to fast deliver the data from the IE to the users. Thus, the users have to subscribe themselves to a topic corresponding to the attribute to be monitored. Successively, any user can get the attribute directly from the topic whenever a new attribute is published by the IE. In this case our measurement campaign is aimed at evaluating the overall traffic present in the network, by varying both the number of the platform users and the frequency at which the attribute changes. It is worth noting that the frequencies of attributes’ renewal have been chosen as the inverse of the average response time estimated when the JMS broker was not active. To better detail this point, let us suppose that under certain specific conditions and with the JMS disabled (i.e., with the system operating in polling mode), the average response time estimated by JMeter is T_{r} and L_{p} is the average traffic load measured over the network. Then, the goal of our measurement campaign is to estimate the network traffic load L_{j} and compare it with L_{p}, when the JMS broker is enabled and $1/T_{\mathrm{r}}$ is the renewal frequency of the attribute handled by the IM.

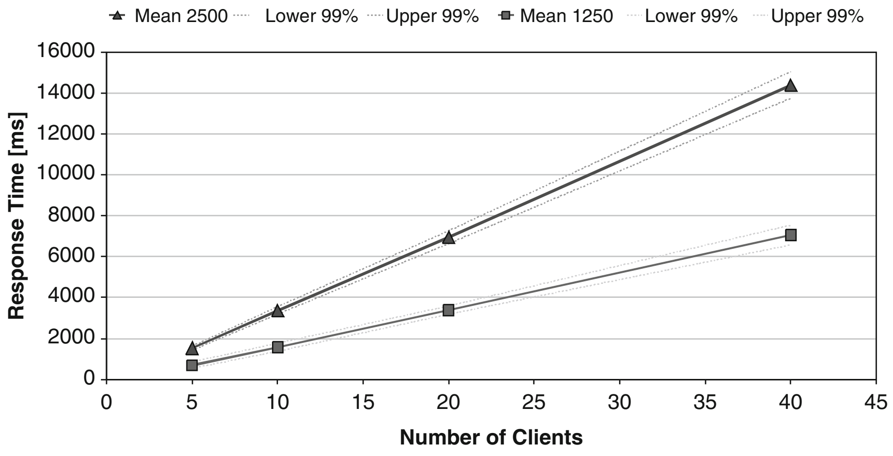

Fig. 3 Response time versus the number of clients when the "Polling Mode" is used

Figure 3 reports the behavior of response time versus the number of clients which poll the IE in order to get a data vector. The upper line (marked with triangles) refers to the case where the IE response consists of a vector of 2,500 doubles (i.e., a vector of 20,000 bytes); the lower line (marked with squares) shows the response time when the IE returns a vector of 1,250 doubles. The two dotted lines on either side of each solid line represent the borders of the confidence interval at 99%.

It is worth noting that: (a) the response time increases almost linearly with the number of clients and (b) the time spent by a requester to obtain a response from the IE could be unacceptable for near real-time applications in most cases, even if only a limited number of users are accessing the laboratory controlled by the IE.

Figure 4 shows the response rate (viz., the inverse of the response time, T_r) versus the number of clients (N_c) when the response of the IE consists of a vector of 2,500 and 1,250 doubles. The solid and dotted curves represent the best polynomial interpolating functions of the measured data.

The interpolating functions are almost hyperboles, thus highlighting that the product $R_r \cdot N_c$ is equal to a constant K_1, which, in turn, weakly depends on the length of the data vector returned by the IE. Observing the data in Table 1 can shed light on the factors affecting the overall IE performance.

Indeed, Table 1 reports the traffic load (kbytes/s) offered to the transmission network by varying the number of clients (a) connected to the IE when the "Polling Mode" is in use or (b) picking data from the JMS broker when the "JMS Subscribing Mode" is enabled.

It should be noted that, when the "Polling Mode" is active, the traffic load does not practically depend on the number of clients accessing the IE: actually, the load is almost constant and equal to about 1 MB/s, i.e., 8 Mb/s. Hence, the IE performance

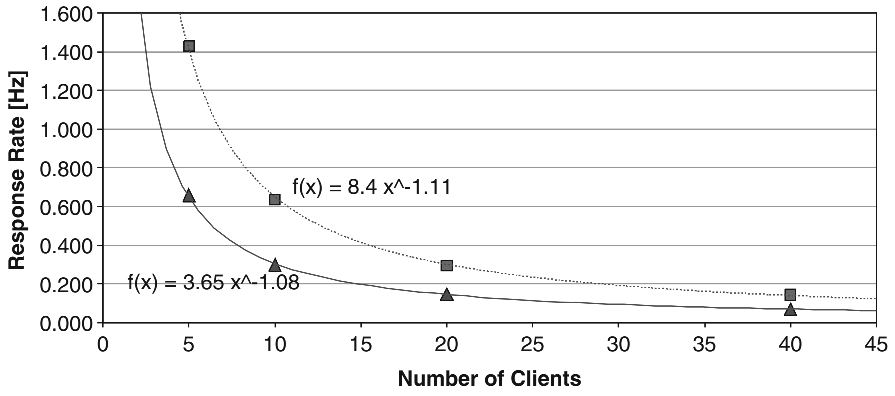

Fig. 4 Response rate versus the number of clients in the "Polling Mode"

Table 1 Average network traffic load (kbytes/s) under different operative conditions

	Number of clients			
	5	10	20	40
Polling 1250	1, 214	1, 063	1, 003	971
JMS 1250	199	161	150	145
Polling 2500	1, 137	1, 027	1, 000	966
JMS 2500	165	154	151	149

is not surely limited by the network capacity (100 Mb/s during our tests), but mainly by the IE's specific implementation, the computational capabilities of the grid node hosting the IE, and partially by the JMeter.

Furthermore, Table 1 presents the traffic load when the JMS broker is exploited by the IE and the IM (of the IE) internally polls the device at a sample frequency equal to the average response rate achievable when the "Polling Mode" is adopted. To better clarify this point, in the "Polling Mode" when five stations are active and the IE produces a response vector of 2,500 doubles (assumed to be the payload of a packet at the network layer), the average response rate amounts to 0.658 vectors/s (corresponding to an average response time of 1,520 ms, see Fig. 3) and the total traffic offered to the network is about 1,137 kB/s (see Table 1). On the contrary, in the "JMS Subscribing Mode," when five stations contemporarily read data vectors (of 2,500 doubles) published, via the JMS broker, by the IM every 1.52 s, the total traffic load amounts to 165 kB/s.

Analyzing Table 1 highlights how the adoption of the "JMS Subscribing Mode" permits us to scale down the traffic load by a factor of about six; alternatively, keeping the traffic load fixed, the IM can sample the device/instrument about six times faster than the case when the "Polling Mode" is adopted. The better performance achievable by the "JMS Subscribing Mode" can be ascribed to two main factors. The first and the most relevant one depends on the different data structures employed by the IE in polling mode and the JMS broker to dispatch data to the clients. As a matter of fact, the IE polling mode exploits the data structures provided by XML/SOAP, which are about 6.7 and 5.7 times more "space"-consuming than the data structures adopted by the JMS broker in the cases of 1,250 and 2,500 doubles, respectively. The second factor, which becomes more evident when a higher quantity of data is managed (viz., in the case of 2,500 doubles), likely depends upon the more efficient implementation of the JMS broker with respect to that of the IE in polling mode. In the case of 2,500 doubles, the data in Table 1 show a ratio of more than 6.48 between network traffic loads produced by the "Polling Mode" and the "JMS Subscribing Mode," while the ratio between the memory space used by the data structures is about 5.7.

In any case, in many situations the benefits resulting from the adoption of a JMS broker may significantly improve the realism perceived by users while handling remote instrumentation.

5 Conclusion

In this chapter, we have evaluated the data gathering performance of the IE, the main software component of the GRIDCC/DORII architecture devoted to remotely control the resources present in real laboratories. During all the tests, the JMeter tool suite has been employed to estimate the time spent by a user to get a measure under the two different operational modes provided by the IE. The measurement campaign revealed that the "Polling Mode" may be often inadequate to get long data arrays from instrumentation, even if only a limited number of users access the laboratory. This mode seems to be more suitable for obtaining atomic, asynchronous data. On the contrary, in case the user requires a continuous monitoring of a data array (e.g., an oscilloscope trace), the "JMS Subscribing Mode" is surely effective, as it allows reducing the total traffic load while increasing the reactivity of the entire platform that manages the remote laboratory. Specifically, the performance tests highlighted that the "JMS Subscribing Mode" permits us to scale down the traffic load by a factor of about six and, conversely, keeping the traffic offered to the network fixed, the user can require the IE to interrogate the device/instrument about six times faster than in the case when the "Polling Mode" is in use.

Future work will be aimed at evaluating the possible use of a JMS broker to gather data from a multitude of small sensor nodes that can be seen as a sort of distributed probe of a complex instrument, whose computation capabilities are inside the IE.

Acknowledgments This work was supported by the European Commission, Information Society, and Media DG, Unit F3 'GÉANT & e-Infrastructure', in the framework of the DORII FP7 Project (contract no. 213110).

References

1. GRIDCC project website, http://www.gridcc.org.
2. RINGrid project website, http://www.ringrid.eu.
3. DORII project website, http://www.dorii.eu.
4. V.J. Harward et al., “The iLab shared architecture: A Web Services infrastructure to build communities of Internet accessible laboratories”, Proc. IEEE, vol. 96, no. 6, pp. 931–950, June 2008.
5. F. Davoli, N. Meyer, R. Pugliese, S. Zappatore, Eds., Grid-Enabled Remote Instrumentation, Springer, New York, NY, 2008; ISBN 978-0-387-09662-9.
6. F. Lelli, E. Frizziero, M. Gulmini, G. Maron, S. Orlando, A. Petrucci, S. Squizzato, “The many faces of the integration of instruments and the grid”, International Journal of Web and Grid Services, vol. 3, no. 3, 2007, pp. 239–266.
7. F. Davoli, S. Palazzo, S. Zappatore, Eds., Distributed Cooperative Laboratories: Networking, Instrumentation, and Measurements, Springer, New York, NY, 2006; ISBN 0-387-29811-8.
8. I. Foster, “Service-oriented science”, Science Mag., vol. 308, no. 5723, May 2005, pp. 814–817.
9. D.F. McMullen, R. Bramley, K. Chiu, H. Davis, T. Devadithya, J.C. Huffman, K. Huffman, T. Reichherzer, “The Common Instrument Middleware Architecture”, in F. Davoli, N. Meyer, R. Pugliese, S. Zappatore, Eds., Grid Enabled Remote Instrumentation, Springer, New York, NY, 2009, pp. 393–407; ISBN: 978-0-387-09662-9.
10. The RINGrid project team, “Remote Instrumentation Whitepaper”, available at http://www.ringrid.eu.
11. L. Berruti, F. Davoli, G. Massei, A. Scarpiello, S. Zappatore, “Remote laboratory experiments in a Virtual Immersive Learning environment”, Advances in Multimedia, vol. 2008 (2008), Article ID 426981, 11 pages, doi:10.1155/2008/426981.
12. L. Berruti, F. Davoli, M. Perrando, S. Vignola, S. Zappatore, “Engineering applications on the eInfrastructure: The case of telecommunication measurement instrumentation”, Computational Methods in Science and Technology, Special Issue on “Instrumentation for e-Science”, vol. 15, no. 1, pp. 41–48, 2009.
13. L. Caviglione, L. Berruti, F. Davoli, M. Polizzi, S. Vignola, S. Zappatore, “On the integration of telecommunication measurement devices within the framework of an instrumentation grid”, in F. Davoli, N. Meyer, R. Pugliese, S. Zappatore, Eds., Grid Enabled Remote Instrumentation, Springer, New York, NY, 2009, pp. 282–300; ISBN: 978-0-387-09662-9.
14. K. Haase, “JMS Tutorial”, available at http://download.oracle.com/docs/cd/E17477_01/javaee/1.3/jms/tutorial/1_3_1-fcs/doc/jms_tutorialTOC.html.
15. “Apache JMeter”, The Apache Jakarta Project, available at http://jakarta.apache.org/jmeter/.