Use of a CORBA/RMI Gateway: Characterization of Communication Overhead

Alessio Bechini, Pierfrancesco Foglia and Cosimo Antonio Prete
Università di Pisa
Dipartimento di Ingegneria dell'Informazione
via Diotisalvi, 2 56100 PISA (Italy)

{a.bechini, foglia, prete}@iet.unipi.it

ABSTRACT

Many distributed applications make use of distributed object technology. In this kind of systems, modules providing services are implemented as objects spread over a network. Distributed objects are usually accessed through communication frameworks based on specific middleware solutions, such as CORBA, DCOM, and RMI. Applications of this kind might be built up (or extended) integrating different modules, possibly already coded and available on the market. Each required and available module might use a specific communication framework, hampering its prompt integration into a system exploiting a different framework. A convenient way to tackle this problem is the insertion of a gateway module, passing service requests between two different middleware solutions. This approach allows a quick integration of service modules, but it could lead to performance problems, due to the introduced communication overhead. In this paper, we report our experience in developing a simple CORBA/RMI gateway module, and we discuss how it affects the application performance. Measures of the communication overhead show that the employment of the gateway is a viable solution in many real-world applications, and gives hints for efficiently placing modules on the available hosts.

Keywords

Performance measurement, distributed object technologies, middleware gateways, CORBA, RMI.

1. INTRODUCTION AND RATIONALE

A number of distributed applications make use of middleware solutions, and particularly distributed object technology (resulting from merging object-oriented techniques with distributed systems technology).
Objects (or modules) providing services are usually accessed by frameworks based on specific middleware solutions. This kind of applications can be built up integrating several modules, possibly already coded and available on the market. Anyway, they might be developed upon a different framework, hampering their prompt integration. There are two possible ways to tackle this problem:

1. Modification of the module source code, changing the communication features in order to make it suitable to interact with another framework;
2. Adoption of an inter-working solution, allowing passing service requests between two different middleware solutions.

The second approach can employ different technologies, with different levels of transparency towards the programmers. The most transparent one is based on the insertion of a gateway module (e.g. in the IONA Orbix COMET, between CORBA and DCOM), specifically generated for each application. The response time for a service call through the gateway is longer than in the case of a direct call, because of the doubling-up of marshalling and un-marshalling operations and possibly of the transmissions on the network. Other inter-working solutions require specific compilation tools, and avoid additional marshalling/un-marshalling overhead, but this way the inter-working architecture is no more transparent to the application developer. A further approach to inter-working between CORBA and DCOM, called "ActiveCOM", has been presented by Daniel, Traverson and Vallee [3].
Although the modification of the module source code is usually much more troublesome, it is likely the most suitable to deliver better overall performances. On the other hand, the insertion of a gateway module allows a quick integration of service modules, but it could lead to performance problems, as previously noticed. Thus, the gateway approach is often more convenient, whenever the additional delay in the communication system does not make the application violate its own timing requirements.
The design choice about the insertion of an inter-middleware gateway should be driven by predictions of the delivered performance [9] [15]. Such kind of predictions can be carried out following a number of diverse approaches, both analytical ([19], [2]) and simulative (e.g., see [6]); in any case, the values for characterizing the corresponding performance models have to be ascertained from actual measures. Thus, the basic behavior of the involved communication software and the gateway module has to be investigated, measuring the timing overhead they introduce in the overall computations.
In this paper, we report our experience in developing a simple CORBA/RMI gateway module, and we discuss how it affects the application performance. We must say that the use of RMI-IIOP [18] could be a good alternative to the gateway. Anyway, the porting process to RMI-IIOP requires not-negligible

WOSP '02, July 24-26, 2002 Rome, Italy

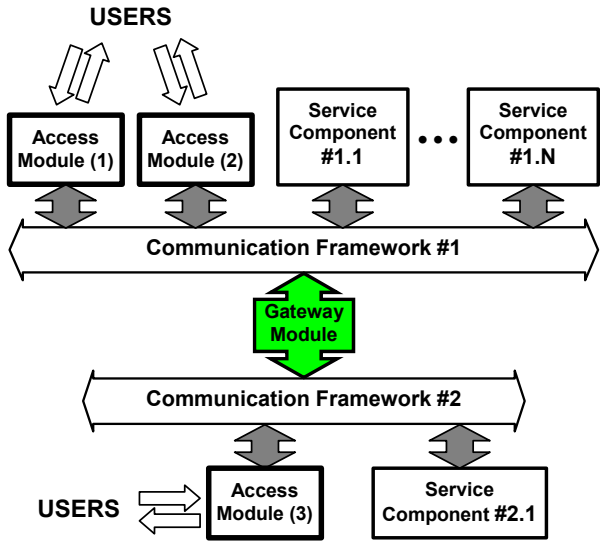


Figure 1 – Use of a gateway module between two different transport frameworks.

modifications to the source code [18], impeding a prompt portability of application modules. Thus, the gateway can still be regarded as a practical and viable solution in realistic situations. Moreover, the similarities between CORBA and RMI make them easily comparable, and allow a simple implementation of a gateway: this is an ideal situation for investigating on their performance, and for understanding the impact on communication overhead of an inter-working solution through a gateway.
The employment of a middleware gateway is particularly appropriate in the field of web applications deployed on a web server system.

1.1 Use in Web Applications

Web server systems are designed to host a number of different web applications. Such applications exploit a set of resources (data and business logic) to satisfy requests coming from users on Internet. Interactions between users and the applications are handled by a pool of dedicated *access modules*, according to the client/server paradigm [11]. The access module of a specific application exploits the functionality of many other modules (known also as *service components*), possibly distributed, providing ad-hoc services, typical of the application domain. Thus, *within* the web applications, access modules often act as *clients* of service components, invoking methods of the objects they are composed of. Especially in the field of e-commerce applications, the architecture of this kind of systems requires a transport framework (offered by proper software technologies, such as RPC, CORBA, RMI, DCOM, etc.) to make every service component accessible from the others. A user request onto an access module triggers a sequence of interactions among the service components in the system.
The increasing demand of new and more powerful services on Internet leads to a need for extending existing web server systems. An extension always affects the overall system performance, by increasing system load and by changing interactions among service components. In order to avoid a significant performance reduction, architectural modifications have to be carefully planned.
A simple solution for integrating components in a web server system is based on keeping the same transport framework and adding upon it a number of service components. In this context, scalability issues can be addressed simply distributing the service components on different computation nodes. As a specific component is particularly stressed, it can be duplicated on different nodes, and each new instance can be accessed by disjoined applications: moreover, it is recommendable to place busy modules on different nodes. Scalability and reliability issues can also be achieved via a clustered service [14].
Another solution is based on adopting an additional transport framework. The introduction of a *gateway module* allows service components on different transport frameworks access each other (Figure 1). In this case, the additional transport framework (based on another technology) can host other access modules and service components, providing additional services to the hosted applications. Although this solution is often deemed convenient, it is important pointing out that all the *critical* services should always be accessed directly, to prevent any possible performance problem.

1.2 Related Works

In the last few years, several works have been presented on performance measurement of applications using middleware support for communications. Some of them focused on inter-working issues.
Inter-working between the two most popular distributed object platforms, i.e. CORBA and DCOM, is an important goal in the industrial field: this is also witnessed by the most recent OMG CORBA specifications [10], defining a co-operation architecture between CORBA and DCOM, based on insertion of a gateway module. Moreover, other kinds of solutions have been proposed, such as Active COM by Daniel, Traverson and Valee [3].
A precise work on measuring and optimizing communication overhead in a CORBA environment has been carried out by Gokhale and Schmidt [5], whose prime goal was to point out improvements making CORBA suitable for latency-sensitive applications over high-speed networks. The authors test the communication latency, taking into account four operation invocation strategies, different number of servant objects and different request invocation algorithms (for addressing scalability issues), and the amount of transmitted data. All the measures are taken for VisiBroker 2.0 and Orbix 2.1.
For distributed Java programs, it makes sense to compare the performance of CORBA and RMI, two similar distributed object models, to understand if one of them is preferable in a given setting. Juric, Rozman and Hericko actually made this comparison [8], and they have shown that CORBA provides much more scalability than RMI, while the communication overhead for both depends on a large number of factors (among the others, the exchanged data size).
Performance analysis of the most important distributed objects models for Java (RMI, remote method invocation, and RMI-IIOP, remote method invocation over internet inter-orb protocol) have been presented by Juric et al. [7]. These authors compared the

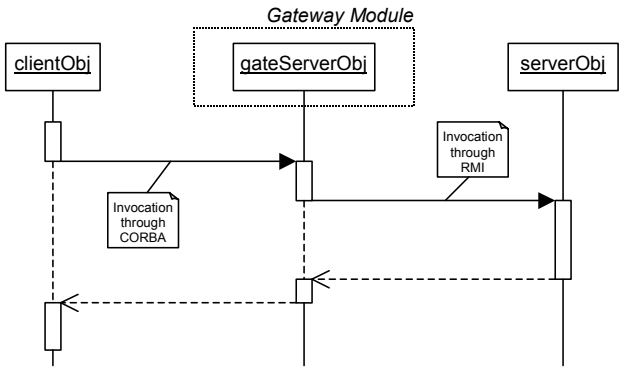


Figure 2 – UML sequence diagram illustrating the logical operations corresponding to the invocation of a method in a remote object through the gateway module (with client on CORBA, server on RMI)

communication overhead in the two models taking into account their behavior with the transmission of primitive data types, data of different sizes and testing also scalability in multi-client scenarios. As expected, RMI-IIOP (that uses the CORBA infrastructure for implementing the distributed method invocations) shows a much longer communication overhead respect to RMI, in every test conditions and especially with large data sizes.

Although it is beyond the scope of this paper, we must say that a sophisticated use of distributed objects in an inter-working scenario requires access also to meta-data through a gateway, and a naïve approach to solve this problem could yield misleading results. A discussion about such issues can be found in a paper by Smith, Gough and Szyperski [16], addressing CORBA/COM inter-working applications.

Smith and Williams have shown how performance models of distributed systems using CORBA middleware can be built [15]: this is a crucial step to design such systems according to performance requirements. A similar approach could be followed also for systems using a gateway, and measures like those presented in our work can be used to infer some of the parameter values in the corresponding performance models.

This paper is organized as follows. Section 2 presents an overview of the structure of the CORBA/RMI gateway module. Section 3 is aimed at presenting the system settings for obtaining meaningful performance measures, and highlights what are the parameters possibly influencing the communication overhead. Section 4 summarizes our actual measures and suggests how to use them in characterizing performance of systems using this kind of communication framework. Section 5 is dedicated to concluding notes.

2. STRUCTURE OF THE GATEWAY MODULE

Applications with client modules on CORBA, and others with clients on RMI require a different implementation of the gateway module. Anyway, the overall structure for both implementations is the same, and can be sketched as follows.

The gateway module is composed of a pool of objects that represent the counterpart of remote objects providing specified services through its methods. Figure 2 illustrates by means of a UML sequence diagram [1] the *logical* behavior exhibited by the gateway in supporting a method invocation on a remote object (serverObj) by a client object (clientObj). The request is first delivered on the gateway to an object (gateServerObj) that is accessed through a given middleware solution (e.g. CORBA). Thus, gateServerObj must be able to communicate by such kind of middleware. GateServerObj is aimed at gathering requests from the client, and forward them to the final recipient (serverObj, known through its exported reference) communicating by another kind of middleware (e.g. RMI). Return values are then passed back, following the reverse path.

On the gateway, gateServerObj can act as a wrapper (or "adapter", following the design patterns' terminology in [4]) towards the reference to serverObj. The wrapper is in charge of passing the actual parameters of the method invocation to the server object, placing them in the form suitable for the middleware used by the final recipient of the service request. All the objects in the gateway, wrappers of the remote server objects, are managed within a *gateway process*, whose main actions are:

i) Lookup of the remote server objects
ii) Construction of the wrappers, passing them the references to their remote server counterparts;
iii) Binding of wrappers to the registry, only in case of client on RMI.

A specific gateway must be generated for each application. The steps to build it are the following:

1) Creation of the wrapper object. This can be done automatically, because a wrapper has a standard structure, and the required information can be retrieved from the interface of the corresponding remote server object.
2) Make the gateway process operate on all the needed wrappers. This can be easily done registering the wrappers into a configuration file for the gateway process.

Usually, each communication framework offers specific facilities to the programmer (e.g. the Dynamic Invocation Interface in CORBA), in addition to the basic ones for managing the exploitation of remote objects. A full-featured gateway should provide a way to deal with such specific facilities, even if they are not particularly important in an inter-working setting. The focus of this paper is the investigation of the *basic behavior* of the gateway, and thus we shall not address explicitly any particular mechanism to deal with specific features.

3. PERFORMANCE MEASUREMENT SETTINGS

Most of the results presented in this paper are derived from the measure of the total time taken by a remote method invocation. Such a measure is commonly known as RTT (Round Trip Time). RTT in distributed systems is a very practical performance index, directly expressing the delay experienced in the invocation. Its role is similar to runtime for applications residing on a single computer [13].

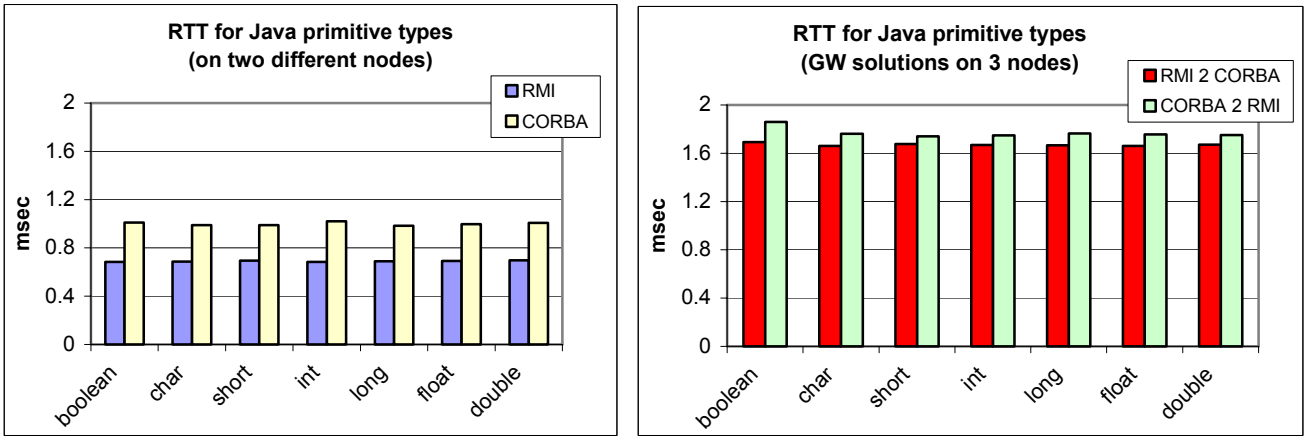


Figure 3 – RTT for obtaining a value of a primitive type. The first diagram compares the RTTS for RMI and CORBA, with server and clients on two different nodes. The second shows the corresponding RTTs in two systems with a gateway.

In our setting, we are interested in the time spent in the communication tasks, regardless of the computational activity on the server side to come to the result. Thus, in our experiments we invoke remote methods that perform no action but possibly return the required value. The actual transmission latency is *usually* a negligible fraction of the whole RTT, because service components are typically placed on hosts connected by a single LAN.
Another possible index to show how the communication overhead is increased by the use of a gateway and an additional middleware framework is the *RTT degradation*, defined as

$$D_{RTT} = \frac{RTT_{gateway}}{RTT_{oneMW}}$$

where $RTT_{gateway}$ stands for the RTT obtained in the communication through a gateway (and two middleware frameworks), and RTT_{oneMW} stands for the RTT on the corresponding request over a single middleware (that one used by the client module).
The whole communication involves many different software activities at different abstraction levels, and likely up to four transmissions of data over a network. In such an intricate structure, there are many different factors that influence the communication delays, from the actual algorithms used in middleware software to the behavior of the transport protocols they rely on. From an abstract point of view, we want to hide as much as possible any detail in the communication framework, pinpointing only its user-perceived performance behavior. Anyway, in building a performance model for systems based on distributed-object technologies, it has been shown that some aspects of the framework must be explicitly modeled [15], such as request scheduling, request dispatching, etc. All these aspects fall out of our investigation, and we focus on measuring the plain communication overhead, trying to understand what are the most important parameters it depends on. Thus, it is important to investigate on the following issues:

1. How different data types as parameters and returned values influence the RTT.
2. How the number of parameters in an invoked method impacts the communication performance.
3. How the data size influences the RTT.
4. The performance degradation (on the RTT) due to the use of a gateway, instead of porting the module upon the main middleware framework.
5. The communication overhead in a typical use within a web application.

The Java primitive types and the corresponding IDL types are reported in Table I; our measures must take into account most of them as type of the returned value in method invocation on remote objects.

Table I

Primitive types in Java and their counterparts in CORBA IDL

Java Primitive type	CORBA IDL type	Length (bits)
boolean	boolean	8
char	wchar	16
byte	octect	8
short	short	16
int	long	32
long	long long	64
float	float	32
double	double	64

3.1 Configurations

Our simple gateway can be used for passing requests from a client operating either in a CORBA or in an RMI framework. Both these two different roles must be tested, as we cannot affirm a priori that both behave in the same way.
We expected that the overall system performance were affected by the placement of modules on the available hosts. Thus we decided to set up the following configurations, depending on the number of used nodes:

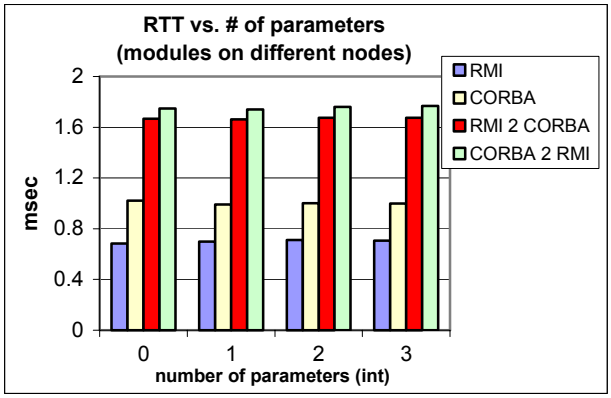


Figure 4 – RTT on invocation of methods with different numbers of parameters; both the argument and returned types are `int`.

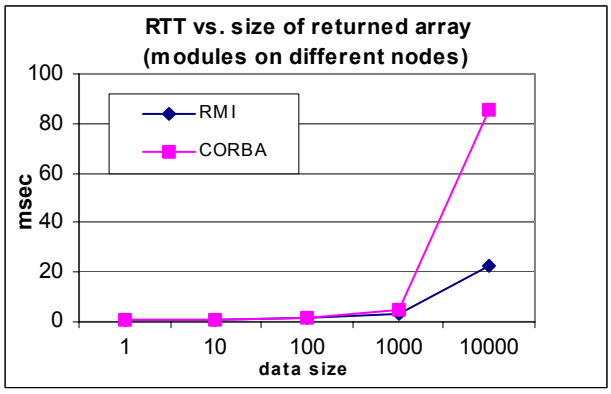
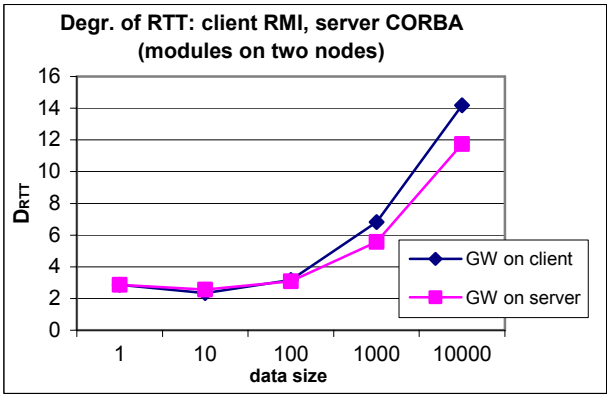


Figure 5 – RTT on invocation of methods which return array of `int` of different size.

1. One single node: client, gateway and server all on the same host
2. Two nodes, with two configurations:
   2.1. Client and gateway on the first host, server on the second one
   2.2. Client on the first host, gateway and server on the second one
3. Three nodes: one for the client, one for the gateway, and one for the server.

Measures have been gathered for each of these configurations, but only some of them are reported in this paper for space limits.

In our work, we didn't plan to measure communication overhead depending on the number of remote objects accessed by clients. This choice is motivated by previously presented results [5], showing that the overhead due to the VisiBroker activity in demultiplexing requests to proper objects on a single node is not dependent on the number of objects (provided that it is less than the maximum allowed). This property comes from the adoption of a demultiplexing strategy with an *O(1)* table lookup algorithm. Thus, it is useless to check it again in our target environment.



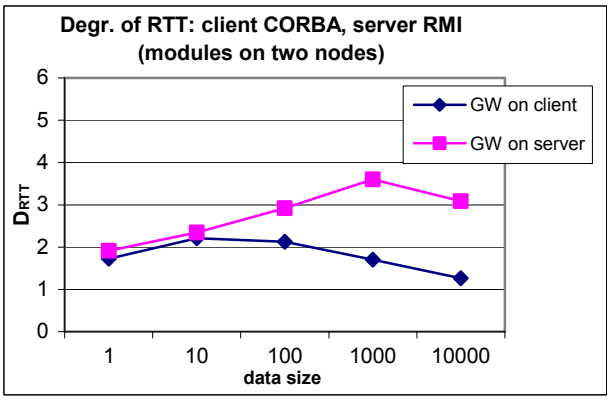


Figure 6 – Degradation of RTT for different placements of the gateway module.

In a multi-client scenario, concurrent requests on the same remote object must be delivered to its unique counterpart on the gateway. Thus, the gateway surely becomes a critical bottleneck. In our specific implementation of the gateway, we didn't apply any particular technique to avoid this behaviour (harmful from the performance standpoint). Anyway, measures on a multi-client scenario (e.g. as those ones reported in [7] and [8]) would be aimed at testing the efficacy of the applied technique. We must say that in accessing essential objects repeatedly requested, the communication overhead must be kept as low as possible, and the adoption of a gateway in this context might not be convenient. Hence, even if it could be interesting to investigate on scalability issues regards to the number of clients, it is not a crucial issue from a practical point of view.

3.2 Software and Hardware Testbed

We carried out our tests with simple applications, all written in Java. The Java source code was compiled and executed within JDK 1.3.0-c. Client applications were run as applets on the JDK's appletviewer. Time was measured properly using the System.currentTimeMillis() static method: in order to collect accurate results, a single measure was obtained repeating 10000 times the corresponding method invocation. The reported results

are calculated as the average of fifteen measures, following the methodology used in [9].
For experiments with CORBA, the Inprise VisiBroker for Java version 4.0 has been used. VisiBroker is one of the most widely used ORBs, compliant with the OMG CORBA specification. All the computers were equipped with a Pentium II 350 MHz with 128 MB RAM, and they used Microsoft Windows NT 4.0 service pack 6. A 100 MBps Ethernet network connected the computers.

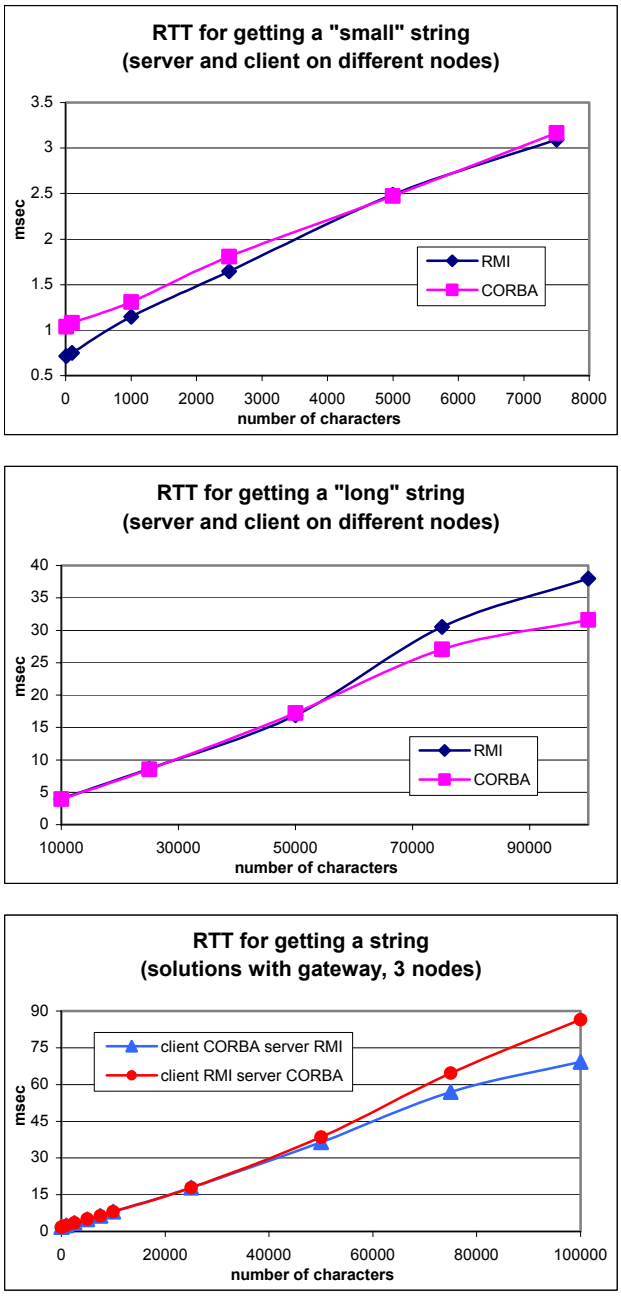


Figure 7 – RTT on invocation of methods returning a string. RMI outperforms CORBA on small strings (first diagram), while CORBA behaves better with very long ones (second diagram). The two solutions with a gateway (last diagram) present similar RTTs with small strings.

4. PERFORMANCE RESULTS

The measured RTT to get a value belonging to Java primitive types is shown in Figure 3, for CORBA, RMI and the two inter-working solutions comprising both of them. The two objects (client and server) are located on two different nodes. Results clearly show that the type of the returned value has no influence on RTT. RMI always provides a quicker delivery respect to CORBA, according to other results obtained in previous works [8]. The gateway insertion (in a configuration with client, server and gateway placed on three distinct nodes) almost doubles up the RTT (Figure 3, diagram on the right). Also in this situation the type of the returned value does not substantially affect RTT. Under this test, the configuration with the client module that communicates through RMI shows quicker RTTs.
We have observed that the number of parameters does not affect the communication overhead in a significant way, both for CORBA and RMI. The gateway doubles up the RTT and we still experience a very low dependence of RTTs on the number of parameters (Figure 4). In our analysis we take into account only a small number of parameters (from 0 to 3 arguments), according to what usually happens in real-world applications.

When dealing with array of different sizes (Figure 5), CORBA and RMI behave in a appreciably different way, because CORBA exhibits a much higher RTT (for array sizes bigger than 1000 elements). This higher RTT value determines the communication overhead (Figure 6) introduced by the gateway between an RMI client and a CORBA server. In this case, the measured D_{RTT} value is 14 for a returned array size of 10000 elements, with an absolute RTT of about 286 msec.
In the invocation of methods returning strings of different length (Figure 7), RMI outperforms CORBA on small strings, while CORBA behaves better with very long ones. The two solutions with a gateway present similar RTTs with small string, with a max D_{RTT} value of 3 for the configuration with a client on RMI and a server on CORBA and 2 for the other.
Finally, it is important to understand how to place client, gateway and server modules on the available nodes. With a three-node configuration, we can place each of them on a separate computer. With a two-node configuration, it is reasonable to place client and server on different nodes, while it is not straightforward to determine if the gateway should be placed close either to the client or to the server.
The configuration with three nodes presents the lower RTT in all the cases considered, both when dealing with large array (Figure 8) and with primitive data type (results not shown). Again from Figure 8, we can see that, in the case of two nodes, it is convenient to place the gateway close to the object using CORBA, regardless of its role in the communication. It is worth noticing that the performance degradation considerably depends on the data size, and thus the choice about the adoption of the gateway should also take into account the amount of data involved in the communication. In the data transmission, the buffering policy adopted by the communication frameworks can determine the saw-tooth behavior of RTT, as shown by the diagrams in Figure 8.

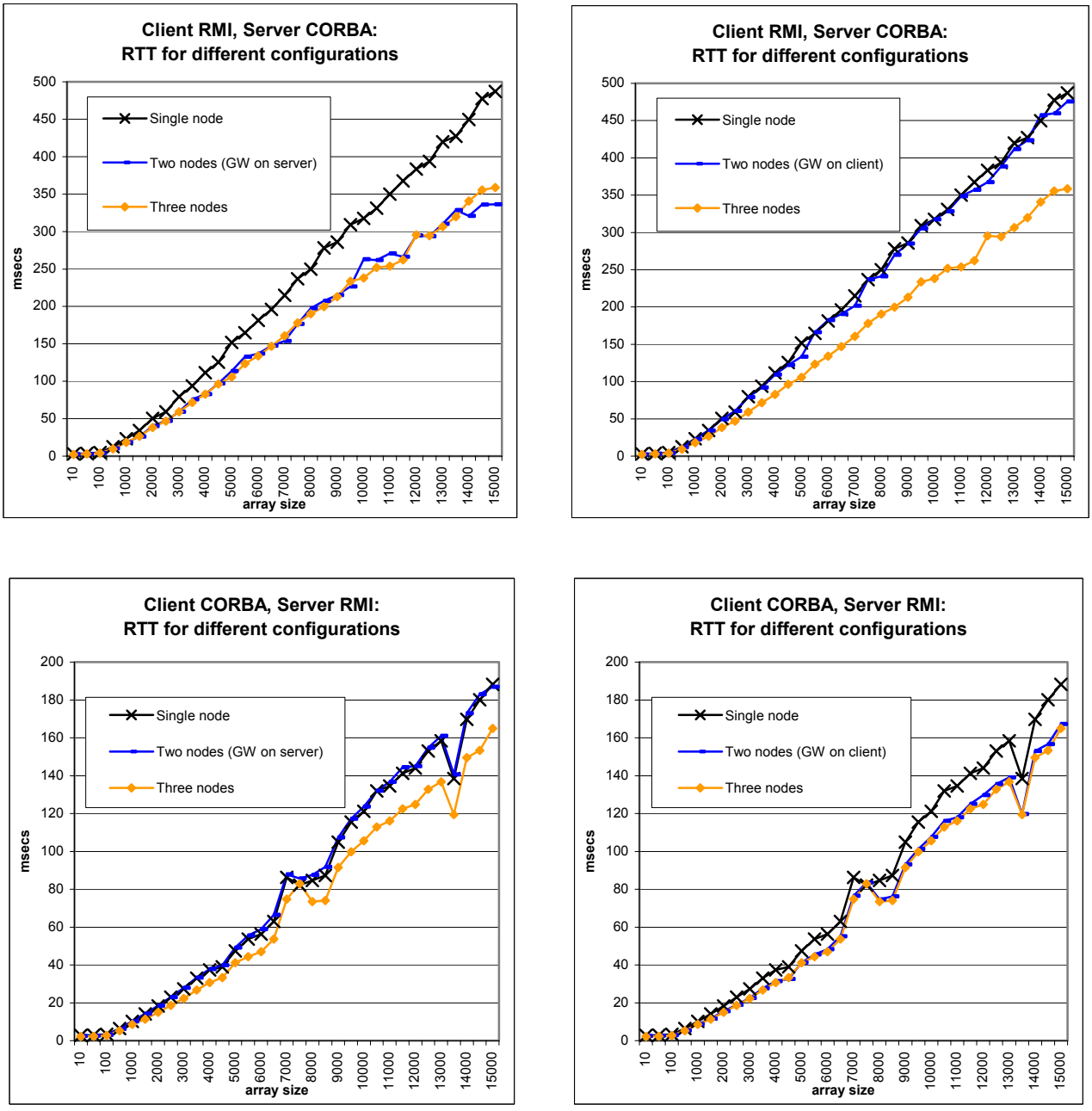


Figure 8– Comparison of RTTs in the two solutions with a gateway, changing the size of an array of integers returned to the client. In all the configurations with two nodes, the lowest RTT can be always obtained placing the gateway close to the module that communicates by CORBA.

Our results can be summarized as follows: the insertion of a gateway between the CORBA and RMI frameworks almost double up the RTT in dealing with arguments of primitive types, and the RTT is approximately insensitive of the number of arguments in the invoked method signature. In these latter cases, the RTT is less than 1 millisecond (including the network latencies). In transferring large arrays, the RTT becomes more and more significant, and it assumes different values according to the placement of client, server and gateway modules on the available nodes. In any case, better results are achieved whenever the modules are spread over two or three nodes: the network latencies are negligible with respect to the time spent in the middleware software activities.

5. CONCLUSIONS

In developing distributed applications, a simple solution for prompt integration of modules communicating by means of either CORBA or RMI is the introduction of a gateway module. This solution determines an additional communication overhead. In this paper, we have presented performance measures aimed at characterizing such overhead. The dependence of round-trip time on different factors has been shown, and this result can give us hints about the possible adoption of the gateway in several specific cases. The performance degradation corresponding to inter-working with two middleware platforms instead of one has been evaluated: it is up to the application developer either to deem it acceptable, or to go on with a more expensive and time-consuming process of module porting onto a unique communication framework. It is important to point out that many distributed applications are equipped with sophisticated GUIs, and the time spent by local CPUs in managing them often "hides" the communication overhead. In this kind of settings, performance degradation due to the gateway frequently does not affect the user-perceived responsiveness of the whole system.

6. ACKNOWLEDGMENTS

The authors wish to thank primarily Giuseppe Monno, whose contribution has been fundamental in the development of the work presented in this paper. Thanks also to the anonymous referees, for their precious comments.

7. REFERENCES

[1] G. Booch, I. Jacobson, J. Rumbaugh. The Unified Modeling Language Users Guide, Addison Wesley, 1998.

[2] D. E. Culler, R. M. Karp, et al. LogP: Towards a Realistic Model of Parallel Computation, in Proc of ACM SIGPLAN Symposium on Principles & Practice of Parallel Programming (PPOPP) 1993, 1-12.

[3] J. Daniel, B. Traverson, V. Vallee, Active COM: an Inter-working Framework for CORBA and DCOM, in Proc. of IEEE Int'l Symp. On Distributed Objects and Applications, Sept. 1999, 211-222.

[4] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. Design Patterns. Addison Wesley 1994.

[5] A.S. Gokhale, D. C. Schmidt, Measuring and Optimizing CORBA Latency and Scalability over High-Speed Networks, IEEE Trans. on Computers, 47(4), Apr. 1998, 391-413.

[6] R. Jain. The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modeling. John Wiley & Sons, 1991.

[7] M. B. Juric, I. Rozman, A. P. Stevens, M. Hericko, S. Nash, Java 2 Distributed Object Models Performance Analysis, Comparison and Optimization, in Proc. of 7th IEEE Int'l Conf. on Parallel and Distributed Systems (ICPADS'00), 2000.

[8] M. B. Juric, I. Rozman, M. Hericko, Performance Comparision of CORBA and RMI, Information and Software Technology Journal, 42(13), Oct. 2000, Elsevier Science, 915-933.

[9] D. A. Menascé, and H. Gomaa. A Method for Design and Performance Modeling of Client/Server Systems. IEEE Trans. on Software Engineering, 26(11) (Nov. 2000), 1066-1085.

[10] Object Management Group, The Common Object Request Broker: Architecture and Specification, Revision 2.6, Dec. 2001.

[11] R. Orfali and D. Harkey, Client/Server programming with Java and CORBA, John Wiley and Sons, 1997.

[12] J.A. Rolia and K.C. Sevcik, The Method of Layers, IEEE Trans. on Software Engineering, 21(8), Aug. 1995, IEEE CS Press , 689-700.

[13] S. Sahni, V. Thanvantri. Performance Metrics: Keeping the Focus on Runtime. IEEE Parallel and Distributed Technology, 4 (1), (Spring 1996), IEEE CS Press, 43-56.

[14] R. Short, R. Gamache, J. Vert and M. Massa, Windows NT Clusters for Availability and Scalability, in Proc. of 42nd IEEE Int'l Computer Conference, San Jose, CA, Feb. 1997, 8-13.

[15] C.U. Smith and L. G. Williams, Performance Engineering Models of CORBA-based Distributed-Object Systems, chapter in book: R. Puigjaner, et.al. Eds. LNCS: Computer Performance Evaluation Modelling Techniques and Tools, Springer, 1998.

[16] G. Smith, J. Gough and C. Szypersky, A Case for Meta-Interworking: Projecting CORBA Meta-data into COM, in Proc. of TOOLS Pacific 1998, Melbourne, Australia, Nov. 1998, 23-26.

[17] M. van Steen, S. van der Zijden, H. J. Sips, Software Engineering for Scalable Distributed Applications, in Proc. of 22nd Int'l Computer Software and Applications Conference (CompSac), Aug. 1998.

[18] Sun Microsystems, RMI-IIOP Programmer's Guide, 1999, downloadable at http://java.sun.com/j2se/1.3/docs/guide/rmi-iiop/rmi_iiop_pg.html

[19] L. G. Valiant. A Bridging Model for Parallel Computation. Comm. ACM, 33(8), Aug. 1990, 103-111.