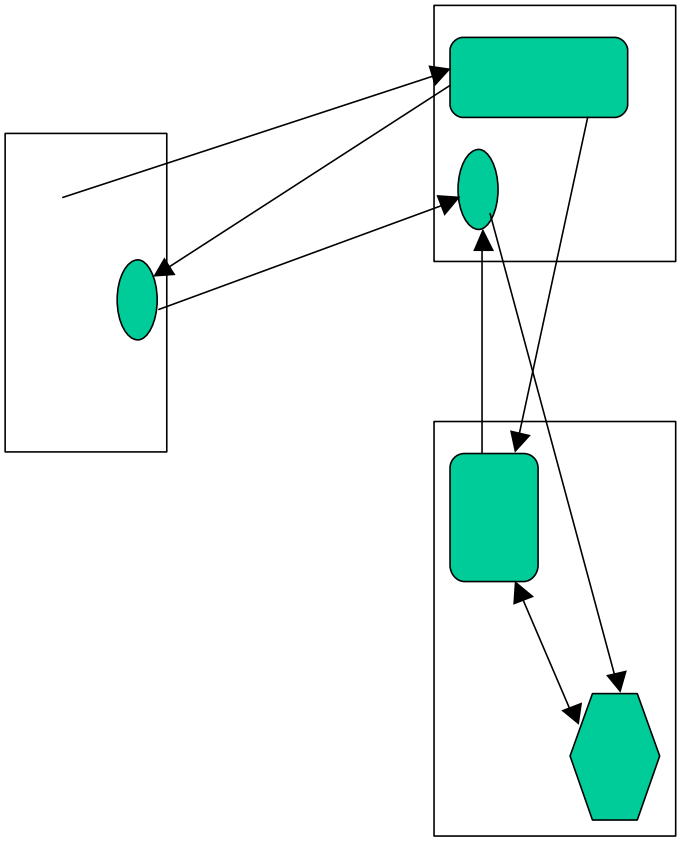
Client and Server Communications of Distributed Systems



OMNI/02/02

Xiuhua Zhang

Oslo

May, 2002

Norsk Regnesentral
ANVENDT DATAFORSKNING

NR-notat/NR Note

Tittel/Title: Client and Server Communications of Distributed Systems

Dato/Date: May
År/Year: 2002
Notat nr: OMNI/02/02
Note no: OMNI/02/02

Forfatter/Author: Xiuhua Zhang

Sammendrag/Abstract:

Distributed systems are any number of workstation servers and hosts tied together via a network that supports any number of applications. This definition covers a broad spectrum of computing trends, client/server, Internet/intranet and distributed object computing architecture. Client/server communications are implemented by different protocols. The discussion of loose coupling between the client and server has been developed around currently available standard and technologies such as RPC, RMI and SOAP as well. Some possible technical solutions based on those technologies are given also. Conclusions at the end provide the considerations for deploying technical solutions.

Emneord/Keywords: Client/Server communication, protocol, RPC, SOAP

Tilgjengelighet/Availability: open

Prosjektnr./Project no.: 636017

Satsningsfelt/Research field: Distributed system design

Antall sider/No. of pages: 18

Norsk Regnesentral / Norwegian Computing Center
Gaustadalléen 23, Postboks 114 Blindern, 0314 Oslo, Norway
Telefon 22 85 25 00, telefax 22 69 76 60

Client and Server Communications of Distributed Systems

Introduction

Distributed systems are any number of workstation servers and hosts tied together by a network that supports any number of applications. This definition covers a broad spectrum of computing trends, client/server, Internet/intranet and distributed object computing architecture. Client/server communications are implemented by different protocols. The discussion of loose coupling between the client and server has been developed around currently available standard and technologies such as RPC, RMI and SOAP as well. Some possible technical solutions based on those technologies are given also. Some conclusions at the end are provided for the considerations in deploying technical solutions of distributed systems on the basis of one example, which, to some extent, offers very common features of distributed systems development with J2EE.

RPC, RMI, and Web Service

It's a specification and a set of implementations that allow software running on disparate operating systems, running in different environments to make procedure calls over the Internet.

1. XML-RPC is a Remote Procedure Calling (RPC) protocol that works over the Internet. XML-RPC to implement clients and servers communication. It is a simple, portable way to make remote procedure calls over HTTP. An XML-RPC message is an HTTP-POST request. The body of the request is in XML. A procedure executes on the server and the value it returns is also formatted in XML.

Procedure parameters can be scalars, numbers, strings, dates, etc.; and can also be complex record and list structures.

How it works: On the wire, XML-RPC values are encoded as XML to make request and send response:

```
<methodCall>
  <methodName>examples.getStateName</methodName>
  <params>
    <param>
      <value><i4>41</i4></value>
      </param>
    </params>
  </methodCall>

<methodResponse>
  <params>
    <param>
      <value><string>South Dakota</string></value>
      </param>
    </params>
  </methodResponse>
```

Supported data types: int, string, Boolean, double, array, datetime:iso8601, base64,

struct.

XML-RPC server side code couldn't be changed and Client side is to be coded in a J2EE application. This could be done by extending XMLRpcClient class or something to be done with parser implementation. Does it has to do something with asynchronous execution.

Strategies/Goals

Firewalls. The goal of this protocol is to lay a compatible foundation across different environments, no new power is provided beyond the capabilities of the CGI interface. Firewall software can watch for POSTs whose Content-Type is text/xml.

Discoverability. We wanted a clean, extensible format that's very simple. It should be possible for an HTML coder to be able to look at a file containing an XML-RPC procedure call, understand what it's doing, and be able to modify it and have it work on the first or second try.

Easy to implement. We also wanted it to be an easy to implement protocol that could quickly be adapted to run in other environments or on other operating systems.

1.1 XML-RPC vs. CORBA

CORBA is a popular protocol for writing distributed, object-oriented applications. It's typically used in multi-tier enterprise applications. CORBA also provides an excellent *interfacedefinition language* (IDL), allowing you to define readable, object-oriented APIs.

Pros

- CORBA is very complex. It requires significant effort to implement, and requires fairly sophisticated clients.
- It's better-suited to enterprise and desktop applications than it is to distributed web applications.

Cons

- To create CORBA based clients that need to access the services of your Java and XML server over the network.

1.2 XML-RPC vs. DCOM

DCOM is Microsoft's answer to CORBA. It's great if you're already using COM components, and you don't need to talk to non-Microsoft systems. Otherwise, it won't help you very much.

1.3 XML-RPC vs. SOAP

SOAP is very similar to XML-RPC. It works by marshaling procedure calls over HTTP as XML documents. SOAP appears to be suffering from specification creep. The initial public release was basically XML-RPC with namespaces and longer element names. Since then, SOAP has been turned over a W3C working group. As of the current writing, SOAP supports XML Schemas, enumerations, strange hybrids of structs and arrays, and custom types. At the same time, several aspects of SOAP are implementation defined.

2 Java RMI

Remote Method Invocation (RMI) is a distributed object model for the Java Platform. RMI is unique in that it is a language-centric model that takes advantage of a common network type system. uses object serialization to convert object graphs to byte-streams for transport. Any Java object type can be passed during invocation, including primitive types, core classes, user-defined classes, and JavaBeans™. Java RMI could be described as a natural progression of procedural RPC, adapted to an object-oriented paradigm.

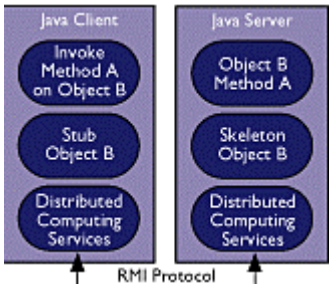


Figure 1. Java RMI Architecture (Source: http://java.sun.com/marketing/collateral/rmi_ds.html)

RMI is implemented as three layers:

- A **stub** program in the **client** side of the **client/server** relationship, and a corresponding skeleton at the server end. The stub appears to the calling program to be the program being called for a service. (Sun uses the term *proxy* as a synonym for stub.)
- A Remote Reference Layer that can behave differently depending on the parameters passed by the calling program. For example, this layer can determine whether the request is to call a single remote service or multiple remote programs as in a multicast.
- A Transport Connection Layer, which sets up and manages the request.

How an RMI Call is packaged within the HTTP Protocol

To get outside a firewall, the transport layer embeds an RMI call within the firewall-trusted HTTP protocol. The RMI call data is sent outside as the body of an HTTP POST request, and the return information is sent back in the body of the HTTP response. The transport layer will formulate the POST request in one of two ways:

- If the firewall proxy will forward an HTTP request directed to an arbitrary port on the host machine, then it is forwarded directly to the port on which the RMI server is listening. The default RMI transport layer on the target machine is listening with a server socket that is capable of understanding and decoding RMI calls inside POST requests.
- If the firewall proxy will only forward HTTP requests directed to certain well-known HTTP ports, then the call will be forwarded to the HTTP server listening on port 80 of the host machine, and a CGI script will be executed to forward the call to the target RMI server port on the same machine.

3 SOAP

SOAP is an XML messaging/RPC standard to enable the exchange of a variety of information. The SOAP library found on the client and the server performs the encoding/decoding. SOAP's encoding/serialization features are mainly used in conjunction with the RPC mechanism. Which may create tighter coupling. The reasons for this include: 1) SOAP client has to know predetermined methods and its parameters, 2) the interface between the client and server and the parameters passed must be known ahead of time by both parties.

Pros
- SOAP appears to be the most widely accepted XML messaging standard;
- Almost all major companies support it.
- SOAP allows both message-oriented or RPC-oriented communications.

Cons
- SOAP requires other supporting standards to provide full functionality.
- Interoperability has not been perfected.
- It's still changing rapidly.

Basic SOAP principles include: open up a HTTP connection, send request in XML format and pass method parameters to invoke a remote method, and read/receive response from the server. There are three basic steps in writing any SOAP-based system:

- Decide on SOAP-RPC or SOAP messaging - RPC or Messaging? RPC systems are simple to write in SOAP and better error handling and passing of complex types across the network. Instead of invoking remote procedures, messaging provides for transfer of information. It's powerful, and doesn't depend on a client knowing about a particular method on some server. It also models distributed systems more closely and complicated than the simpler RPC-style.
- Write or obtain access to a SOAP service - a class that is going to have its methods invoked remotely.
- Write or obtain access to a SOAP client (see below)- programs that invoke SOAP-RPC.

An RPC Client (http://www.oreilly.com/catalog/javaxml2/chapter/ch12.html)
- There are some basic steps you'll take in every SOAP-RPC call:
- Create the SOAP-RPC call
- Set up any type mappings for custom parameters
- Set the URI of the SOAP service to use
- Specify the method to invoke
- Specify the encoding to use
- Add any parameters to the call
- Connect to the SOAP service
- Receive and interpret a response

It can be concluded that a server and a set of clients that just need to perform tasks remotely, then RPC is probably well suited for your needs. However, in larger systems that are exchanging data rather than performing specific business functions on request, SOAP's messaging capabilities may be a better match.

4 Web service

The primary reason to create any Web service is to make software libraries available to clients of all languages at once. It tightly integrates with traditional CGI applications and allows for rapid prototyping.

Web services mean different things to different people. A WebService is any service that handles requests at a well known location such as a URL. In the context of SOAP and Java, Web services are based on SOAP and use a WSDL file to define them. WSDL files are being adopted on both the client and server portion of SOAP communication to better describe the type of communication that will occur between the two parties.

For instance, a WSDL file may describe that a particular message is sent as input to a SOAP server and a particular message is sent in response to that input. A WSDL file also describes at which URL and port a particular Web service exists.

Web Services can overcome integration problems across different systems running across different platforms. In the data interchange process XML can be used to integrate applications by minimizing effort previously required for data transformation across different platforms. Moreover, it's easily extensible, scalable, and highly portable.

WDSL

Web Services Description Language (WDSL) is a new specification to describe networked XML-based services. It provides a simple way for service providers to describe the basic format of requests to their systems regardless of the underlying protocol (such as Simple Object Access Protocol or XML) or encoding (such as Multipurpose Internet Messaging Extensions). WSDL is a key part of the effort of the Universal Description, Discovery and Integration (UDDI) initiative to provide directories and descriptions of such on-line services for electronic business.

UDDI

UDDI is a standard for dynamic lookup, binding, and publishing of SOAP services. It allows you to query different UDDI registries to look up businesses, information by business category, and service information. The UDDI API is divided into two logical parts: 1) Inquiry API: Provide programs with the capability to locate candidate businesses, Web services and then services, and drill into the specifics based on overview information provided in the initial calls. The Inquiry API functions are exposed as SOAP messages over HTTP. No authentication is required to make use of the Inquiry API functions. 2) Publisher API: Enables programs to save and delete each of four data types supported by UDDI. Authenticated access is required to use the Publisher API, i.e. the user must first sign-up with one or more operator sites to establish user credentials.

So what does combining UDDI with SOAP buy you over SOAP alone? One of the areas where XML messaging has been lacking is in the area of directory services. In Java, directory services through JNDI play a crucial role in publishing and looking up various types of services (such as EJBs, JMS queues, and so on). One of UDDI's goals

is to provide for SOAP a service similar to what JNDI provides for Java, all in a B2B scenario.

Scenario to use message-oriented approach
- The client/server communication takes place over the Internet and the client and server belong to different companies
- Developing an event-based system in which events are created and then consumed by interested parties. Most GUIs are event-based. This helps to remove the dependency between an event (or action in a system) and the system's reaction to the event that is performed on the server.

The general components of Web services can be roughly categorized as follows:
- Service Consumers. Service consumers are end users and devices on the service grid, including attended and unattended devices; for example, people using computers, cell phones, PDAs, and ATM machines.
- Service Interface. The service interface implements connection and communication APIs and protocols, transforms and models data for the desired output device, and aggregates, personalizes, notifies, and locates content.
- Service Container. The service container implements the service runtime environment, persistence services, and state management.
- Service Platform. The service platform is the system or device on which services are run. Functions of the service platform include database access, messaging and directory services, virtual machines, operating systems, hardware interfaces, and storage.
- Business Logic. Business logic is the application code that implements business services.
- Service Integration. Service integration functions include integrating other Web services, legacy systems, and infrastructure services such as databases, files, and directories.

The current Web services model requires building protocols, interfaces and products that support this instantaneous combining and recombining of components through:
- Discovery that discovers other relevant services by means of ebXML Registry and Repository and/or UDDI.
- Creation that creates XML content, and develop business logic code.
- Transformation that maps to existing XML, Java or SQL applications.
- Building that wraps components with XML message interfaces, using ebXML Message Service or SOAP.
- Deploying that transmits services to service deployment machines.
- Testing that tests applications as they run locally or remotely.
- Publishing that advertises the service to an ebXML or UDDI registry.

Sun's JAX Pack includes APIs geared for both approaches: JAX-RPC (Java API for XML-based RPC) for RPC-style Web services, and JAXM for document-style services. This loosely coupled approach can even make it possible to develop new application behavior in a declarative rather than procedural way. The more XML messages say about themselves, the less their senders and receivers need to know about one another's infrastructures..

The ability of any of software to be web-enabled. Since XML is web enabled, and your

systems might rely on XML formatted input (and generate XML output), you can easily make these services available on the web by using Servlets. This is a very important feature because XML and Java give you the ability to easily, quickly, and cheaply web-enable your systems, which is not the case if you use other technologies like CORBA.

Java Community RPC – RMI

Java community has developed its own standard and protocol to implement RPC referring to standards such as XML, SOAP and HTTP as well. More detailed information can be found in slides 1-10 in the appendix.

1. RMI over IIOP (Internet Inter-ORB protocol)- A simpler solution than CORBA Interoperability
+RMI uses it's own protocol and more transparently integrated with Java.
+IIOP must be used if a Java program wants to interact with another CORBA program.

2. XML-based RPC

Neither CORBA/ORB or Java RMI has been designed to work with XML based RPC. Where practicable, this JSR should attempt to align with this existing RPC work. The goal of this JSR is to develop APIs and conventions for supporting XML based RPC protocols in the Java platform. There are three main needs to be addressed:

- APIs for marshaling and unmarshaling arguments and for transmitting and receiving calls. These APIs should permit the development of portable "stubs" and "skeletons".
- APIs and conventions for mapping XML based RPC call definitions into Java interfaces, classes, and methods. The purpose of this "forward mapping" is to allow XML based RPC interfaces that have been defined in other languages to be mapped into Java. It is highly desirable to be able to map all XML based RPC call definitions into Java.
- APIs and conventions for mapping Java classes, interfaces, and methods into XML based RPC call definitions. The purpose of this "reverse mapping" is to allow programmers to define APIs in Java and then map them into XML based RPC. There may be some constraints on which Java methods can be mapped into XML based RPC.

3. XML messaging

XML messaging means that the client is more independent from the server, as called loose couple. In general, a message-oriented approach proves more flexible than an RPC approach.

Pros
- Loose coupling
- Easier message routing and transformation
- More flexible payload (can include binary attachments, for example)
- Can use several messages together to invoke a given procedure

Cons

- It requires more work to develop a client/server interaction using a message-oriented approach compared with an RPC approach like RMI because developing a client/server interaction via a message represents another level of indirection from RPC. Complexity is added through the creation of the message on the client side (versus a procedure invocation in an RPC approach) and on the server side with message-processing code. Because of its added complexity, a message-oriented design can be more difficult to understand and debug.
- There is a risk of losing type information for the programming language in which the message was sent.
- In most cases the transactional context in which the message was created does not propagate to the messaging server.
- Scenarios to use messaging:
- Client/server communication takes place over the Internet and the client and server belong to different companies.
- An event-based client/server system is developed. Most GUIs are event-based.

4. XML-based SoapRMI

The design objectives of SoapRMI 1.1 are as follows:

- To implement an RMI system in C++ and Java that conforms to SUN's Java RMI API.
- Provide access to Directory services based on RMI registry and LDAP.
- Architect a SoapRMI server to function as a web-service if needed.
- Automate the generation of stubs and skeletons from XML-Schema based specification of interfaces.
- Enable multiple object serialization mechanisms in SoapRMI-Java and recursive serialization in SoapRMI-C++.
- Use the dynamic proxy classes provided as part of Java 1.3 to reuse the same stub and skeleton for every remote object.
- Support exceptions across C++ and Java platforms in an seamless manner.
- Interoperate with existing SOAP based frameworks like .NET and Apache SOAP.

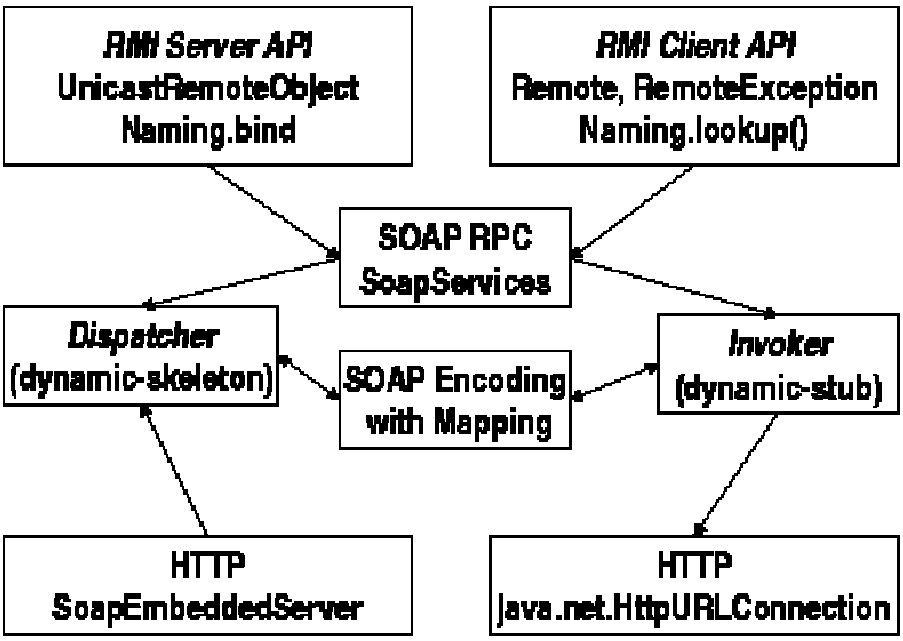


Figure 2. SoapRMI1.1 Java Architecture

Case Study - SIL

1 About SIL

SIL is a system developed by J2EE technology. Because of the confidential reason, SIL is used only as a name to mean the technical solutions to the system like SIL. SIL is willing to achieve the following objectives:

- Loose coupling between client and server
- Both synchronous and asynchronous communication available solutions to having internal clients and external applications
- Application server independent Java implementation (portability)
- High security concern for transaction data
- The development of an open and standards-based network-based application.

2 Alternative solutions to achieve a loose coupling between client and server

Loose coupling means high abstraction and independence between client and server. Client and server could be developed without predefined relations, but the dynamic/runtime ones. Based on the above SIL objectives, four technical solutions are provided and given in Slides 11-33 in Appendix.

SOAP enabled client design. Currently there are many tools available like GLUE in Java (http://www.themindelectric.com/products/glue/example.html). With GLUE, the client only needs to bind to web service at specified URL and invoke the web service as if it was a local object after the server starts up a web server (to accept message) and publishes an instance of the class to be received by the client.

To some extent, Web service solution can be regarded as a host publishing tool (http://eai.ebizq.net/leg/havart_1.html). These tools enable organizations to leverage legacy applications and data in a Web-based environment. Host publishing allows quick access to critical data by providing browser-based access to host applications without requiring any development or changes to existing systems.

3 Open Issues

- The remote programming model like XML and SOAP provides location independence and flexibility with regard to the distribution of components in the deployment environment. It provides a loose coupling between the client and the beans on the server side.
- If SOAP is used, then SOAP client library including XML storage and SOAP server have to be considered.
- Location allocation could be another reason for the tight coupling.
- Import packages/classes are far more than what they used. This is very common in the code programs.

Conclusions

The following conclusions can be gained according to the above discussion.

1. Java RMI is a tight coupling solution, whereas SOAP and XML provide more loose solution. XML is the key enabler to provide this loose coupling.
2. The choice between XML and SOAP may lie in the security consideration. SOAP could provide more secure solution.
3. SOAP could be combined with Java RMI technology. Web services will communicate with each other by exchanging messages, expressed as XML documents.
4. All are about the protocol standards: CORBA, RMI, XML, SOAP. Only RMI is language-specific.
5. Architecture should be flexible enough to handle multiple protocols if it turns different protocols coexisting. What aspects of architecture are they relevant to?
6. If the Java client is written in Swing, then we still have to deal with distributed computing issues in access the object model (on the server) over the network. You might use RMI, IIOP (CORBA) or HTTP tunneling (using Servlets).
7. XML messaging makes any kind of mappings for the applications, because XML development also involves using certain APIs, such as SAX, DOM, Servlets, Swing, RMI, JDBC, and the core Java API.
8. Messaging uses messages to communicate with different systems to perform some kind of function. Messaging today doesn't necessarily mean that you are using a MOM product, and it doesn't necessarily mean that you are communicating asynchronously. Rather, messaging can be either synchronous (two-way) or asynchronous and use many different protocols such as HTTP or SMTP, as well as MOM products. It could be referred to the communication as being message-oriented because you would send and receive messages to perform the operation, in contrast to an RPC-oriented communication.

References

Java RMI http://java.sun.com/j2se/1.3/docs/guide/rmi/spec/rmiTOC.html

Java WSDP, http://java.sun.com/webservices/webservicespack.html

XML messaging http://www.javaworld.com/javaworld/jw-06-2001/jw-0622-xmlmessaging2_p.html

Design of an XML based Interoperable RMI System: SoapRMI C++/Java 1.1, by Slominski et al. (2002), http://www.extreme.indiana.edu/xgws/papers/soaprmi_design/

IIOP-ORB, http://www.cs.rit.edu/~ats/do-2001-1/html/skript-20.html

XML-RPC, http://jcp.org/jsr/detail/101.prt

Appendix: Overview of technology and technical solutions