Using Java For Dynamic Access to Multiple Object Models

Bela Ban
IBM Zurich Research Laboratory
8803 Rüschlikon
email: bba@zurich.ibm.com

October 23, 1996

1 Introduction

CORBA [OMG95] and Java [Sun95] are ideal combinations for creating distributed systems. Java is a platform independent language, suitable to be downloaded to any machine and with its GUI class library, especially suitable for GUI development. CORBA is more geared towards distributed object oriented processing of information. In my opinion, it will be feasible to create future distributed applications using Java for the GUI and CORBA for the model (after the model-view-controller paradigm [GR89]).

If we follow the MVC paradigm, implications are that a view accesses the model, never vice versa. Therefore a mechanism has to be provided that lets Java applications access CORBA instances as if they were native Java classes.

One mechanism developed by the OMG is the provision of a Java language binding and a corresponding IDL-Java compiler. The compiler would translate IDL interfaces to their corresponding Java stub classes and IDL types to equivalent Java types. The operations of the Java stubs would use IIOP to forward the request to a CORBA instance in a CORBA server and convert the result to a Java type. This conversion code is generated by the compiler at compile time.

A disadvantage of this approach is that any CORBA interface a Java application needs to access has to be converted to a corresponding Java class before it can be used. This constrains the nature of Java applications to be rather static. There is no possibility to dynamically discover what CORBA interfaces are available in certain traders / servers and to create instances of them and manipulate them at runtime. This feature would be essential e.g. for Java-based CORBA interface browsers, CORBA interpreters or dynamic 'topology' applications (for management purposes).

The following approach shows how this restriction can be removed. The main advantages are:

- Dynamic access to CORBA interfaces at runtime.

- Smaller client applications. Only little compile time knowledge (fixed number of interfaces) is needed (GOM interfaces `GenObj`, `Val` (and subclasses) and `Factory` [Ban95]).

- Access to other object models. This is quasi a 'free' add-on feature of the GOM architecture itself.

2 Architecture

The architecture is shown in fig. 1.

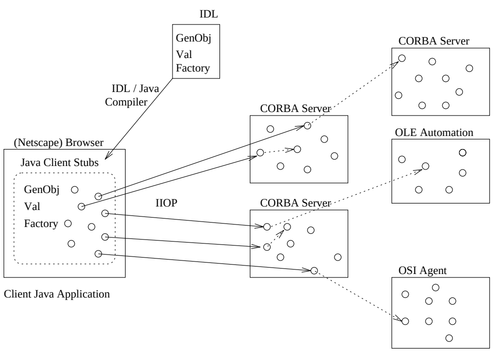

Figure 1: Java / GOM Architecture

The GOM interfaces `GenObj`, `Val` (and subclasses) and `Factory` (for creation of new `GenObj`s) are translated from IDL to their corresponding Java stubs using the proposed OMG IDL-Java translation mapping. These interfaces are the only interfaces that need to be translated; no other interfaces will ever need to be translated because of the reified nature of GOM that uses metadata to manipulate interfaces / classes dynamically.
A request sent to a `GenObj` stub instance will be transparently forwarded to a `GenObj` implementation object in a CORBA server which then uses an adapter to dispatch the request to a specific object model (such as CORBA, CMIP [CMI], OLE [Bro94, Box95], SNMP [CFSD90]etc.).
Given the availability of GOM CORBA interfaces and their implementation and an IDL-Java translator (to be furnished by an ORB implementor), we just need to generate Java classes from GOM's CORBA interfaces and can immediately use them in Java applications without any further interaction. If desired, *convenience bindings* for Java can be created which adapt the generated Java classes to the Java environment and make them easier and more comfortable to use[1].

3 Applications

The type of applications that can be developed using such a model is essentially the same as if GOM was used directly. The main difference is that a Java language binding would be available and that CORBA objects could be accessed from it. This leads to a number of advantages: first of all, Java employs garbage collection which decreases memory management complexity.
Second, the virtual machine architecture of Java enables portability and allows to run a

[1]Convenience bindings are described in my thesis at `http://www.zurich.ibm.com/~bba/diss.ps`.

Java program on any machine without modifications. Thus, Java applets can be loaded on the fly from any location to the local machine and subsequently display the GUI of an application. The (business) logic of the application can be located in CORBA objects either local or remote to the application. Employing Java for GUI purposes has the advantage that the Java code to be downloaded can be quite small which results in fast download times, whereas the model code can nevertheless be accessed using the Java-GOM stub classes for CORBA.

Third, client applications don't need any compiled-in knowledge of classes / interfaces because of GOM which keeps them small in size and decreases the download time of applets. Also, conversion between GOM and a specific object model is done in adapters located in a CORBA server so that this code needn't be downloaded either.

And finally, Java applications needn't be regenerated any time classes / interfaces are added or modified since GOM uses runtime metadata to know about them.

Two further sample applications are presented below.

3.1 JavaScript and GOM

The interpreted language JavaScript in conjunction with GOM would allow to create and manipulate objects from any model interactively. This combination could become an alternative to HTML + CGI e.g. for data access from Web browsers. Data (object-oriented databases) in the form of CORBA objects could be presented using HTML + JavaScript. The latter allows to handle any Java classes, therefore also the generated `Genobj`, `Val` and `Factory` classes.

3.2 A Java Class Inspector

A special type of application useful especially in development environments is a Java inspector that allows to browse classes, create instances and manipulate them.

Since the inspector would be built using the Java stub classes for GOM, it would be able to list classes / interfaces of all object models (e.g. CMIP, CORBA, OLE, SNMP etc.) that GOM supports. Instances of any class could be created on the fly - without any client knowledge about the class' type - attributes of the created instance could be modified and retrieved and operations could be invoked, *interactively* providing arguments.

These features would e.g. allow a CMIP developer to point his Netscape browser to the MIB that he just developed, list all GDMO templates in it, instantiate managed objects in an agent and performing GET-, SET- and ACTION-requests interactively. This allows for rapid testing of behavior (conformance tests)[2], documentation purposes (MIB browsing) or quick discovery and navigation of MIBs not developed in-house.

[2] Using GOMscript, conformance tests can be automated. More information about GOMscript is available at `http://www.zurich.ibm.com/~bba/gomscript.html`

4 Surfing CORBA Objects Across The Web

Currently, the situation of the Web can be compared to that of structured programming in the seventies: information (data) and behavior (programs) are separated; programs are used to manipulate data and the associations that determine which program is to be invoked for which datum[3] are maintained in a file using MIME. This brings with it all the disadvantages of structured programming which we won't deal with here.

Second, the information in the Web is untyped (types as used in programming languages)[4] and have no inherent structure in them, most of the information available on the Web is in the form of ASCII or binary files.

Third, access to the distributed information space is an ASCII-based protocol (HTTP).

It is the author's assumption that these three points may change in the future as follows.

First, the Web may be transformed from the 'structured programming' philosophy to an object-oriented paradigm in that data and programs are encapsulated together, yielding autonomous data entities which have a behavior; that is, they know how to manipulate themselves. For advantages of an object-oriented approach over a structured one refer to respective literature.

Second, information in the Web therefore may become typed, that is, instances of classes.

Third, HTTP as a protocol may be superseded with either IIOP as an 'object-protocol' on the protocol level, or, better, with an RPC-like CORBA API, which is not concerned about protocols, but offers a higher level of abstraction.

In a line, in the future the universal request for information in the form of:

`http://www.zurich.ibm.com/~bba/gom.ps`

may be replaced by:

`orb://com/ibm/zurich/ban/papers/gom`.

Information in the Web may be *specified* using CORBA interfaces and *instantiated* by creating instances of those interfaces. The above request typed within a browser would therefore e.g. call a CORBA naming service to request an object reference to an instance of a paper called `gom` in the hierarchical naming context `com/ibm/zurich/ban/papers`.

CORBA objects can be categorized into two categories: those that know how to display themselves within a container and those that don't. The paper instance retrieved above would have to know how to display itself in the browser's context, otherwise it could not be displayed (maybe only the data members would be shown). But, of course, any 'displayable' object may use other nongraphical objects to fulfill its task. This line of reasoning contrasts somehow with what has been said before; namely that Java will be used for display chores (GUI). But as Netscape has announced[5], their browser will be made into an OpenDoc[6] container, therefore being

[3] E.g. which viewer has to be started when an audio file is received by the browser.

[4] MIME associations can barely be called types.

[5] `http://www.netscape.com`

[6] `http://www.cilabs.org`

able to display OpenDoc parts (constructed on top of CORBA). This offers an alternative to Java for the GUI so that there will be a choice between the two (and probably more) for GUI implementation.
If this vision of a 'typed' Web will become true, then the need to access *objects* distributed across the Web will arise. It is impossible for a browser to know all the types (interfaces) of all the objects it will ever encounter during a session. It is envisaged that a few frequently used types such as files, images, audio and video data, directories etc. will be known by a browser, but the whole universe of types available on the Web cannot be known.
Therefore, a model based on the concept of generic types / classes and runtime instance creation / operation dispatching using metadata as exemplified by GOM will be needed to handle information on the Web. The association between an instance that represents a piece of information and its type which is located in an interface repository can be compared to that of a piece of information and its MIME 'type' as determined in a file in the Web server. But unlike information in the current Web, 'object-oriented information' is structured in the form of classes and have behavior.

References

[Ban95] Bela Ban. Extending Corba For Multi-Domain Management. Technical report, IBM Research Division, IBM Zurich Research Laboratory Saeumerstr. 4 8803 Rueschlikon, September 1995.

[Box95] Don Box. Building C++ Components Using OLE2. *C++ Report*, 7(3):28–34, April 1995.

[Bro94] K. Brockschmidt. *Inside OLE2*. Microsoft Press, Redmont, WA, 1994.

[CFSD90] J. Case, M. Fedor, M. Schoffstall, and C. Davin. The Simple Network Management Protocol. RFC 1157, May 1990.

[CMI] International Standards Organization. *Common Management Information Protocol (CMIP)*. ISO / IEC 9596.

[GR89] Adele Goldberg and David Robson. *Smalltalk-80. The Language*. Addison-Wesley, 1989.

[OMG95] *The Common Object Request Broker: Architecture And Specification*. July 1995. Revision 2.0.

[Sun95] Sun Microsystems. *The Java Language Environment: A White Paper*, 1995.

Bela Ban is a researcher at IBM's Zurich Research Laboratory, working on multi-object model management problems.