CORBA Application Development Environment Using Reflection

Kazuhiro Fujieda
School of Information Science,
Japan Advanced Institute of Science and Technology

March, 2000

Abstract

CORBA is the standard of the Object Request Broker (ORB), the middle-ware to realize distributed object environment. CORBA is independent of platforms (operating systems and hardware architectures) and programming languages. There are several implementations of CORBA on various platforms from embedded hardwares to mainframes. Distributed objects and their clients in CORBA can be implemented with various programming languages including non-object-oriented languages. CORBA ORB ensures clients and distributed objects to work together regardless of their platforms and their implementation languages.

On developing CORBA applications, developers must describe the specifications of distributed objects by the Interface Description Language (IDL), and generate program codes called stubs or skeletons with the IDL compiler, and then incorporate them into the applications. This development procedure leads large development effort, and often causes interface mismatches between servers and clients when they are developed on different platforms. This paper presents solutions of these problems with the use of the Interface Repository and programming languages supporting the reflection.

The Interface Repository is the server in the meaning of CORBA providing storage of specifications described by IDL. Developers can share it for sharing the specifications on various development platforms for the sake of platform independency of CORBA. The development environment with the Interface Repository can prevent interface mismatches on cross-platform development. The environment consists of the Interface Repository improved to record the modification time of its contents, the editing tool for it, and the tool managing the consistency between stubs/skeletons and the specifications stored in it. This environment doesn't depend on a specific programming language or a specific platform, so developers can use it on various style of application development combining programming languages and platforms.

If developers use the programming languages supporting the reflection to develop applications, the dynamic stubs/skeletons generator using the descriptive power of the reflection can simplify the development procedure. It automatically generates and incorporates stubs/skeletons necessary for the applications at runtime. The specifications necessary to generate them can be

retrieved from the Interface Repository. The descriptive power of the reflection is categorized into the linguistic reflection and the behavioral reflection. The former allows a program to inspect and modify itself. It realizes the runtime generation of stubs/skeletons. The latter allows a program to inspect and modify the behavior of the runtime environment of itself. It is useful to detect the stubs/skeletons necessary for applications and to incorporate them into their programs.

The runtime generator using the linguistic reflection and the Interface Repository can generate only necessary portions of stubs/skeletons apart from the syntax of IDL or the syntax of the target programming language. Moreover the linguistic reflection allows to partially modify incorporated stubs/skeletons when the corresponding specifications are modified.

By means of the behavioral reflection, the generator can detect and incorporate stubs/ skeletons necessary for applications without any extra programming effort or development procedure. There are two kinds of hints for detecting them: the runtime behavior of the applications and the runtime behavior of the execution engine of them.

The object references obtained by clients is useful as the former hint. The runtime library of CORBA linked with the clients can detect the necessary stubs by applying the standard operation for obtaining the interface specification to an object reference. It can invoke the generator with the obtained specification, and incorporate the generated stubs into the client program by manipulating the runtime name space through the behavioral reflection.

An undefined name detected by the execution engine of applications is useful as the latter hint. The generator can modify the behavior of the execution engine when the program refers an undefined name through the behavioral reflection so that the resulting value of the name becomes the generated stub or skeleton corresponding to it. With the use of the undefined method, the generator can generates the special stub methods for using APIs such as the CORBA Messaging only when they are necessitated by applications.

This paper presents implementation of this approach with the Python and the Java which support the reflection and have implementations of CORBA. The Python provides the descriptive power of the reflection enough to implement all of the approach. The runtime generator implemented with the Python can use object references obtained by clients and the behavior of the Python interpreter in referring undefined module and method names to detect necessary stubs/skeletons. The Java provides only narrow power of the reflection, so the generator can generates stubs/skeletons only class by class, and can't modify generated classes. The generator implemented with the Java can use the behavior of the Java virtual machine in referring undefined classes to detect necessary stubs/skeletons.

Either implementation can automatically generate and incorporate stubs/skeletons necessary for applications, so the development procedure becomes much simpler. These languages are often used to implement only the client side of applications. When the server side are implemented by other languages, the development environment using the Interface Repository mentioned above can prevent interface mismatches. Especially in the prototyping phase of application development, the specifications of distributed objects are often modified, so the environment and the runtime generator can considerably eliminate development effort.