NISTIR 6202

Conformance Testing Object-Oriented Frameworks Using JAVA

Kevin G. Brady
James St. Pierre

U.S. DEPARTMENT OF COMMERCE
Technology Administration
National Institute of Standards
and Technology
Electronic Information Technologies
Group
Gaithersburg, MD 20899-0001

NIST

NISTIR 6202

Conformance Testing Object-Oriented Frameworks Using JAVA

Kevin G. Brady
James St. Pierre

U.S. DEPARTMENT OF COMMERCE
Technology Administration
National Institute of Standards
and Technology
Electronic Information Technologies
Group
Gaithersburg, MD 20899-0001

July 1998

U.S. DEPARTMENT OF COMMERCE
William M. Daley, Secretary

TECHNOLOGY ADMINISTRATION
Gary R. Bachula, Acting Under Secretary
for Technology

NATIONAL INSTITUTE OF STANDARDS
AND TECHNOLOGY
Raymond G. Kammer, Director

Conformance Testing Object-Oriented Frameworks Using JAVA

Kevin G. Brady and James St.Pierre
Electronic Information Technologies Group
National Institute of Standards and Technology
Gaithersburg, MD 20879

Abstract

This paper details the assumptions, decision processes, and conclusions reached during the development and implementation of a Conformance Testing Tool using the JAVA **[refer:JAVA]** programming language, and the Object Management Group's (OMG's) Common Object Request Broker Architecture (CORBA) specification **[refer:CORBA]** for distributed object communication. JAVA was used to implement the tests, and CORBA was used to provide the communication interface between the JAVA tests and the object under test. This test tool is being continually refined and expanded by the National Institute of Standards and Technology (NIST), but it was initially conceived as part of a collaborative effort between NIST and SEMATECH, 1994 to 1997, in support of the development of the SEMATECH Computer Integrated Manufacturing (CIM) Framework **[refer:CIMF]**. SEMATECH [1] had the foresight to include plans for conformance testing at the early stages of the CIM Framework specification development, and asked NIST to assist in this task. Although this initial implementation was developed for the semiconductor industry, the concepts, software and conclusions are relevant and applicable to conformance testing of any object-oriented framework.

Introduction

The programming language "JAVA" and its relationship to the World Wide Web (WWW) is familiar to most people. Although most people are familiar with the name,

---

[1] **Specifically, Alan Weber, project leader for the SEMATECH CIM Framework.**

they are unfamiliar with what JAVA actually is and what it actually does. Never before has a computer language been developed that has been so widely accepted and implemented **[refer:JAVAFACT]**. JAVA is an object oriented programming language, whose primary difference from other object oriented programming languages is its ability to run on any computer platform that supports the JAVA Virtual Machine (JVM). The JVM is the software which is embedded in web browsers to allow them to run JAVA applets - software programs written in JAVA. The most exciting aspect of JAVA is its potential to allow programmers to “write once, run anywhere.” This goal has not yet been fully achieved, but with the amount of industry and research resources being applied to the problem it is expected that it could be achieved in the near future. JAVA is riding on the coat-tails of success the WWW has experienced. The ability to run JAVA applets is now found in almost all web browsers. It is interesting to note that many people do not realize that the browser they use to “surf the web” contains a JAVA Virtual Machine. [As people surf the web they encounter JAVA applets all the time, and some may not even realize that they are running an “applet.”] Software conformance testing methodologies have always been faced with the difficulty of providing testing tools that are portable across a wide variety of heterogeneous platforms **[refer:NIST]**. The widespread availability of the JAVA programming language makes it an ideal candidate for use in conformance testing.

SEMATECH’s CIM Framework Specification is a document which defines an application framework for Computer-Integrated Manufacturing (CIM) within semiconductor factories. A framework is a software infrastructure that creates a common environment for integrating applications and sharing information in a given domain. While originally designed to support semiconductor manufacturing, the design of the CIM Framework does not prohibit its extension into other manufacturing domains **[refer:CIM]**. The CIM Framework specification describes the object abstractions, attendant services, and message protocols necessary to build compatible applications for the semi-conductor manufacturing floor. Because the CIM Framework is defined as a collection of objects (using the OMG’s Interface Definition Language (IDL)) which can communicate via OMG CORBA technology, it readily lends itself to the use of an object oriented programming language, such as JAVA, for conformance and certification testing.

Conformance Testing - Myths and Methodologies

Conformance testing in general has two major beneficiaries; the application user and the application developer. The application user, or organization, benefits from the assistance conformance testing provides to the creation of procurement specifications. When a user wishes to purchase software, conformance testing provides a means for them to "measure" the level of conformance of a software application to the specification (often an ANSI/ISO standard). The user can then procure software with a greater understanding of the completeness of the application (i.e., conformance). This information can also be used as a requirement specification in the selection process. Thus, for the user, the completeness of the application can be directly measured.

The application developer, or vendor, benefits from the availability of a conformance test tool. As an application is developed, conformance tests can be run against the individual "pieces" to ensure correctness, before they are integrated to form larger applications. Correcting errors at the earliest possible point in the development process is cost effective for vendors. Developers do not have to wait until they have completed a software project to begin running the conformance tests. Issues as simple as "inconsistent interpretation" of a specification can be eliminated from the software development process by having conformance tests available at the earliest possible cycle of the process. By following this methodology the developer can have a high degree of confidence that the software product will be conformant when it is completed.

In trying to develop a test suite before an actual implementation is available, checking the correctness, depth, and breadth of the test suite becomes very difficult. Most developers of conformance tests use alpha and beta versions of a vendor's implementation to exercise the test suite, hoping to flush out any major errors in the tests being developed. [The reason most conformance test suites are so very far behind the development of the actual software is that they are waiting for something to test; at the same time the vendors need tests to verify their implementation code.] This creates a classic "chicken and egg" problem. In an effort to address this problem, researchers at NIST are developing a software tool that will allow test developers to simulate application software, so that the test suite can be developed prior to the existence of any commercial implementations. The Manufacturer's CORBA Interface Testing Toolkit (MCITT) currently under development in the Manufacturing Engineering Laboratory

(MEL) at NIST, will be able to simulate objects given a description of their behavior **[refer:MCITT]**. This testing tool will be used in the next phase of this project when we develop more of the actual test suite. So that the test suite can be verified, the MCITT reads a file containing Interface Testing Language (ITL) definitions and generates skeletal objects that respond correctly to inquiries. As a simple example, assume an object has a name parameter and a status parameter. The test tool would read the definitions and automatically create an object that replies with a character name and the correct status, but does not have to implement the functionality of the object. The test suite then requests, and subsequently verifies, the name and status from the object. The availability of such a test tool allows conformance tests to be developed well in advance of the actual implementation. One advantage the test tool has over an actual implementation is that faults can be programmed as well, something actual objects can not do (e.g., simulate failures of objects). Tools of this nature will make conformance testing and development of the software they test much more independent, decoupling the dependence of one upon the other.

Conformance Testing - Object Oriented Style

As stated above, the goal of this project was to develop a prototype implementation demonstrating the process of conformance testing a working implementation of the SEMATECH CIM Framework. The CIM Framework is an Object-Oriented (OO) specification, and therefore posed additional considerations not found in most simple applications. Chief among those considerations was the "language" to use to implement the test suite. Since it is an Object-Oriented application, an OO programming language was a primary requirement. Until a few years ago, the choices were very limited and C++ would most likely have been the language of choice. But with the many different "flavors" of C++, multiple copies of a test suite, compiled for different hardware platforms would have been necessary. With the advent of the WWW and the JAVA programming language a new choice was available. The pros and cons of each of these languages will be discussed later in this paper. The widespread availability via web browsers and the platform independence of JAVA make this language particularly useful for conformance testing. As we shall see, distribution of code, version control, running of the test suite and certification are greatly simplified using JAVA and CORBA.

Interface Definition Language (IDL)

In order for two pieces of software to interact, a common interface must be defined. Usually, for non Object Oriented software, this interface is simply a set of subroutine calls with the number and types of each parameter strictly defined. As an example, consider a graphics library. To draw a simple line a user would call a subroutine “line” with four integer parameters. The four parameters would represent two sets of X and Y coordinates, the start and end points of the line to be drawn. Each routine in the library would have a specified number of parameters and each of a specific type (e.g., character, integer, etc.). In the Object Oriented (OO) world, interaction is a more complex issue, because each object not only has attributes of specified types, but can contain its own set of function calls that operate on those attributes. A more robust system for defining objects was required. The Interface Definition Language (IDL) is the unifying bond that most object oriented frameworks share. It allows the framework developer to express the application interface completely, so that two different implementations can interact. The SEMATECH CIM framework IDL describes the attributes and behaviours for each of its objects. The implementor builds each object according to the specification, with a common interface to the object instances. Most object-oriented languages contain a special compiler for IDL to assist the implementor. The compiler parses the IDL and creates “stub” files (e.g., skeleton files) for both the client and server applications. The programmer provides additional code to implement the object behaviours and inserts it into the skeletal routines created by the compiler. This ensures that the application interface remains exactly as specified, with no errors in the definitions of the object’s interface.

The Object Management Group (OMG) is the standards body responsible for IDL. The OMG does not develop standards, rather they endorse implementations that become standards. The IDL definition for JAVA will soon be a standard, and will be the last piece of the conformance testing puzzle to fall into place. Once the IDL definition is complete, JAVA applications will soon begin to flourish and surpass their peer C++ implementations (at least in client applications).

Interface Definition Language (IDL)

In order for two pieces of software to
Usually, for non Object Oriented
calls with the number and types of
consider a graphics library. In the
with four integer parameters. The
coordinates, the line speed and
would need to specify the
integer,
be
of function
objects
ma
application
CELL
objects
com

Common Object Request Broker Architecture (CORBA)

Another piece of the puzzle was solved by using CORBA for communications. Writing the test suite as a JAVA applet introduced certain security restrictions of JAVA. A JAVA applet cannot open a connection to any computer except the computer (web server) it originated from. An applet down loaded from a web server cannot open a connection to the users local machine. This restriction is easily removed if the user down loads the code and runs it as a JAVA application on their computer. However, another option is possible using CORBA and the Object Request Broker (ORB) on the web server. The JAVA applet can connect to the ORB running on the web server, and with information provided by the user, that ORB can then make the connection to the user's ORB using the Internet Inter-Orb Protocol (IIOP). CORBA defines the standard protocol for the two ORBs to communicate (See Figure 1). The server ORB then simply relays all test requests to the client ORB, and relays back the results to the JAVA applet. This simple connection prevents the user from having to down load the software and run it as an application that has no security restrictions.

JAVA vs. C++

C++ has been in development for years and lacks only one real quality that JAVA possesses, widespread implementation. Technically speaking, JAVA is a subset of C++, so there are things you can do in C++ that cannot be done in JAVA (e.g. pointers, multiple inheritance, etc.). JAVA started out as the "Oak" programming language developed by SUN Microsystems for use in the consumer electronics industry. Oak was intended to address a problem that new semiconductor chips presented. In the consumer electronics industry, chip designs were evolving very rapidly, and new software had to be written for each chip, compiled and loaded. The idea was to develop an architecture in which software could be "down loaded" to any new chip. Since it was interpretive, the new code would not have to be compiled for each chip, thus saving precious time and money. When the WWW came into existence, it initially only served HTML pages to browsers. The JAVA development team quickly realized it could also "serve" programs to web browsers in much the same way that chips were being loaded. JAVA quickly developed the needed functionality to integrate with the web and is currently implemented directly in most web browsers. This gives JAVA the wide-spread availability that C++ does not have, making it

Common Object Request Broker Architecture (CORBA)

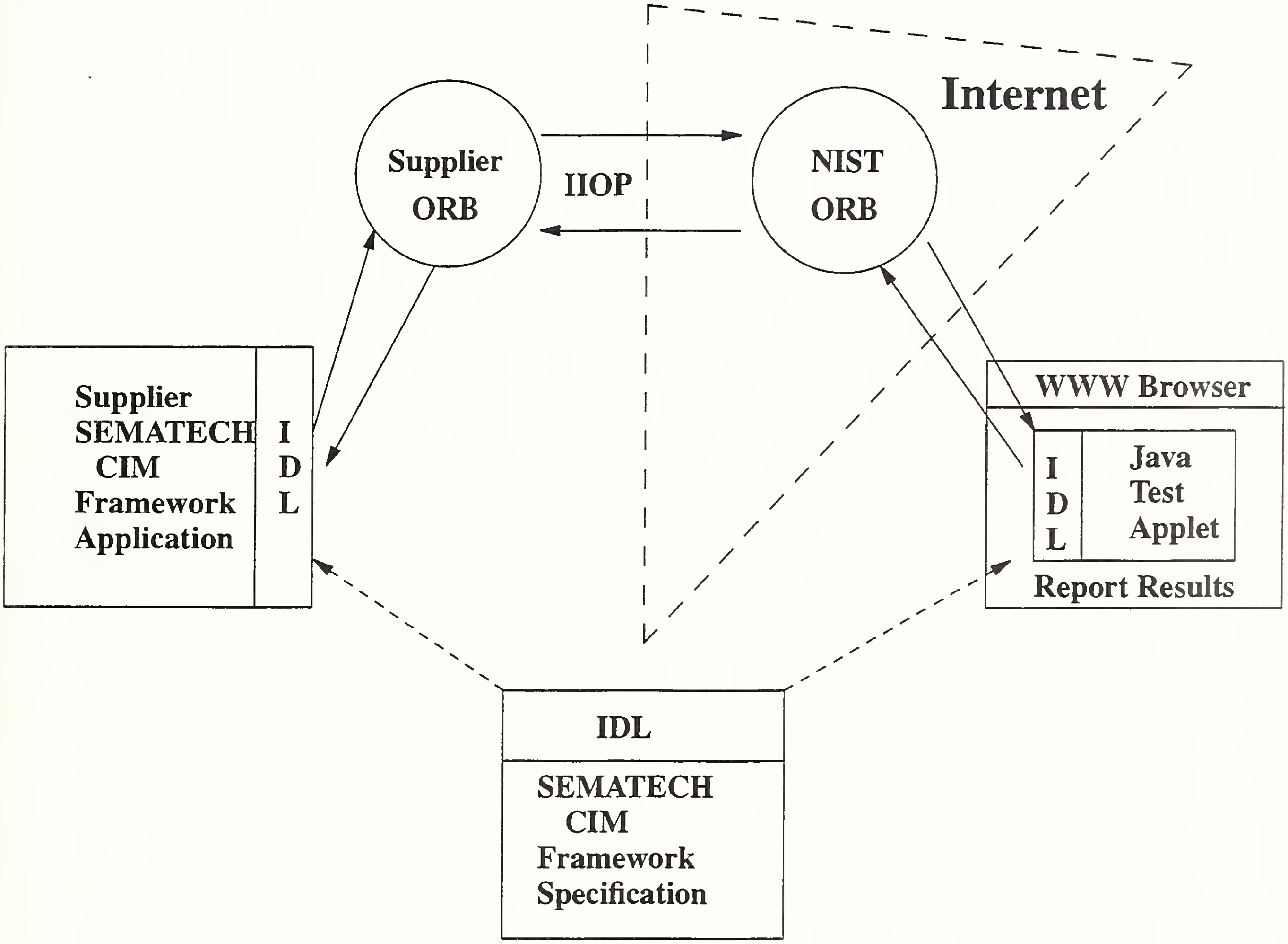
Another piece of the puzzle was [illegible] Writing the test suite as JAVA applet introduces [illegible] JAVA applet cannot [illegible] it originated from. Applets downloaded [illegible] the user local machine. [illegible] and runs [illegible] using CORBA [illegible] Object Request Broker [illegible] can connect to the [illegible] test [illegible] (IIOP) Protocol [illegible] communicate [illegible] client [illegible] Provide [illegible] this [illegible]

JAVA [illegible]

pe [illegible] de [illegible]

[illegible]

Figure 1. Conformance Testing Environment Using Java and CORBA



a more desirable programming language for the delivery of conformance tests.

C++ will probably remain the language of choice for implementing server side code (the actual functionality of the objects). In conformance testing, JAVA is sufficient to query the objects under test. The IDL compilers exist to assist in writing the client side tests, allowing the tester to instantiate an object and test all of its attributes and functions.

Graphical User Interface (GUI)

Perhaps the most beneficial aspect of the JAVA language, in our opinion, has turned out to be the Abstract Windowing Toolkit (AWT). The AWT is built-in to JAVA, and therefore contained in most browsers. The AWT provides comparable functionality to the X windowing system, allowing the creation and deletion of windows, buttons, graphics, etc. When developing a conformance test suite in the past, the use of any GUI was impossible. No single windowing package was built for all systems, and therefore, anything used could not be assumed to be available on all systems. Hence, the test suite was distributed on all types of magnetic media (e.g., disks, tapes, etc.) or left in a directory for anonymous ftp access in many formats (e.g., tar files, zip files) for workstations, PCs, and mainframes. A document had to be written explaining how to install, run, and interpret the results of the test suite. Now, with JAVA and the AWT, a user interface can be written to step the user through the running of the test suite with the click of a mouse, and the results can be analyzed and displayed by the GUI in any appropriate format. By simply distributing a Universal Resource Locator (URL) of your test suite, the user can load it via his browser. With the click of a mouse, you have distributed the software in a single format, installed it, and as the AWT interface runs, it steps the user though the process of executing the test suite and interpreting the results. Results of the test can be displayed graphically, and the final results displayed (pass or fail), without the user having to analyze any of the results. Conformance certifications, which in the past required an on-site visit to verify the integrity of the test suite source code, can be done remotely. Applets can be signed using internet security procedures with digital signatures, authenticating the user and validity of the source code. Since the code is distributed “read-only”, no on-site presence is required to guarantee source code integrity.

The JAVA test tool provides an ideal harness for conformance testing. The only thing that must change from application to application are the actual conformance tests. The AWT portion can be developed and mostly re-used for each software test suite developed. Source code can be down loaded for viewing, help files distributed, and descriptions of the tests can distributed via the WWW using HTML instead of writing and distributing a paper document. Version control problems for test suites are eliminated because each time a user runs the test suite, they are down loading the latest versions of the tests.

The JAVA Test Tool

With all the pieces in place, the development of the JAVA/CORBA-based test tool could begin. The first step was to build a small reference implementation. Since, as mentioned above, no commercial implementations existed, and the NIST MCITT test tool was not yet available, the class *Resource* and *ResourceManager* from the CIM framework were implemented. The IDL definitions were run through a C++ IDL compiler, to generate both client and server stub files. Then, the functionality in the server module, and a client to *drive* the server, were implemented. Again, these two were written in C++ to verify the IDL implementation (See the source code at the URL referenced below).

Next, when the C++ client and server were interacting correctly, a JAVA-based client to *drive* the C++ server was developed. The same IDL was put through a JAVA IDL compiler to generate the JAVA client stubs. The client code was then developed in JAVA and tested to verify that a JAVA client could interact with a C++ server through the ORB. Since most of the interaction occurs in the ORB and is not visible to the user, a small JAVA thread was developed to graphically display the creation and deletion of the objects as they occur. This gives the programmer visual confirmation that the objects are indeed created, and is a true necessity for demonstration purposes.

Once we had verified the interaction of JAVA clients and C++ servers, it was time to develop the actual test harness and test suite. The test harness implemented as a JAVA applet, and the ORB, must reside on the web server. The GUI development had one major goal - make the running of the test suite as effortless as possible for the user. (This ease of use was achieved by having the user simply supply the address of the machine where the application under test resided. The test tool then queries the ORB on his machine for available servers. Upon selection of a server, with one more click of the mouse we could then run the entire test suite against his implementation. The results of tests are displayed graphically. Each of the 105 classes tested is listed, with green indicating conformant results and red indicating failed results.

For the classes that failed, the first thing an implementor would like to do is view the individual tests that were run against that class. The implementor is permitted to load the individual class, and then run the subset of tests pertinent to that class. Again, results of the tests are displayed graphically, listing the individual tests that passed in green and

failures in red, along with the test numbers. With the click of a mouse, he can view a master list of tests for this object, listed by test number, with a brief description of the purpose of the test. After selecting the test that failed, he can then view the actual JAVA source code of the test, to allow him to see exactly how the test was constructed. The test can be re-run individually, so changes he makes to his implementation can be tested immediately, without having to redo tests that have already passed. This methodology is continued until all tests have passed.

The last part of the conformance testing process would be the certification of the implementation. This normally would have required a site visit, but the next part of this project will outline the security required to perform the certification remotely. With proper security measures in place, the vendor could be issued a certificate detailing the level of conformance to the standard being tested.

Summary

The JAVA language is widely used and implemented, and provides a viable language for use in conformance testing. In conjunction with the OMG's CORBA, it can provide a very effective harness for implementing and delivering a test suite for various applications. Our experience to date has demonstrated its usefulness in the Object-Oriented Frameworks domain. Its applicability to other areas will only be limited by the ability to implement (or drive) an application via a JAVA programming interface. (e.g., this project was made possible because there was a JAVA IDL compiler; if it did not exist we would have had to implement in a language with an IDL compiler like C++). We are looking into other possible applications for this conformance test tool, including testing of computer file formats used for electronic circuit descriptions. The tool would analyze an input file to make sure it was compliant to a given standard. This is analogous to the numerous HTML syntax checkers currently in operation. Given the URL of a particular HTML document (web page), these tools access the page, conformance test it to the HTML standard and report back errors. The one downfall of this HTML testing method is the processing load put on the web server, since all scripts are run on the web server. JAVA would run on the client machine, thereby greatly minimizing the network load, and by using the JAVA AWT a much more robust and user friendly GUI could be developed.

This project is an ongoing endeavor, with NIST's Electronics and Electrical Engineering Laboratory (EEEL) continuing to expand the scope and applications of the test tool to support the electronics industry. Other NIST researchers are also evaluating the test environment for possible use in their domains. The tool has functioned well and can be viewed at the following URL (all source code is also available):

http://megavolt.eeel.nist.gov/CIM.html

From this URL you can control the running of a partial test suite against a prototype CIM Framework application. The test tool is connected to an ORB on a web server here at NIST. In addition, the source code is available if you would like to explore the use of this test environment for your own conformance testing needs. All that is required to run this demonstration is a WWW browser which includes the JVM.

We are expanding the test tool to incorporate the two ORB scheme mentioned above, and are writing more individual conformance tests. We will be exploring the use of Digital Signatures to allow an applet to directly connect to a machine other than the web server. This will lift the restriction that forces the applet to connect to the NIST ORB first and then to the user ORB. In addition we are collaborating with industry to explore further development and commercialization of the tool. The MEL MCITT tool will also be used to simulate some of the objects we are testing.

References

All Web references (URL's) were verified for existence in February 1998.

[CIM]
Web reference [*http://www.sematech.org/member/division/fi/cim/cimhome.htm*]

[CIMF] *Computer Integrated Manufacturing (CIM) Framework document, Version 2.0, Austin, TX:SEMATECH.*
Web reference [http://www.sematech.org/public/docubase/abstract/1697jeng.htm]

[CORBA] Object Management Group, *Common Object Request Broker: Architecture and Specification, Revision 2.1.* Framingham, MA: Object Management Group, 1997.

[JAVA] Web reference [*http://www.sun.com/java/*]

[MCITT] Web reference [*http://www.mel.nist.gov/msidstaff/flater/mcitt/index.html*]

[NIST] James A. St.Pierre, Kevin G. Brady, S.L. Stewart, *Conformance Testing and Specification Management* NISTIR 5879 (1996).

[JAVAFACT]
Sandeep Singhal, Binh Nguyen, *The Java Factor*, Communications of the ACM June 1998 volume 41.