Science Opportunity Analyzer™ (SOA):
Everything Including the Kitchen Sink

R. Jay Torres
Jet Propulsion Laboratory / California Institute of Technology
4800 Oak Grove Drive
M/S 301-250D
Pasadena, California 91109

November 21, 2003

Abstract

Science Opportunity Analyzer (SOA) assists scientists in planning a space mission through parameter computation and visualization. SOA is not your typical “run-of-the-mill” Java application since it requires a language that provides powerful APIs and must be multiplatform as well. SOA would also push the limit of some of these APIs. The APIs include used by SOA include: the Java Native Interface (JNI), Java3D, Java2D, Remote Method Invocation (RMI), and Reflection. This number and complexity of APIs in one package has provided SOA with several unique challenges that had to be resolved.

SOA combines many different APIs into one single application. JNI is used so that the Java portion of SOA can access a complicated navigation library, in C, that performs coordinate transformations written by experts. In visualization, the decision to use the Java APIs for graphics has allowed SOA to take advantage of Java3D’s scenegraph structure and Java2D’s simple yet powerful API. Reflection plays an important role in SOA in that it allows different Space Missions to add their specific components to SOA “on-the-fly” without having to recompile the application. RMI is used in SOA to allow a client to access a complex, third party mathematical software application. In general, these APIs give the programmers the ease of developing in Java and not having to worry about different standards for different languages on different platforms.

Integrating the various APIs has not been as simple as it sounds. Error trapping techniques for using JNI have created various issues including reporting errors. Using Java3D and Java2D also has some drawbacks including memory leaks and slow performance in early releases. Another major problem stems from having to visualize many objects that are very far away. Reflection was a new concept to the programmers at the start of the development phase requiring time to fully understand the API. Getting these APIs together made the software more complex. The difficulty of tracing null pointers, memory leaks, and plain old bugs in using the APIs proved overwhelming at times. There were countless sleepless nights that, in the end, resulted in finding ways to overcome these obstacles.

Some tools such as, Sun’s jvmstat, freeware JSwat, Rational’s OptimizeIt, and Quest Software’s JProbe suite have been used with varying results. Though they provide assistance to many problems, they are also lacking in debugging some of the APIs including Java3D and JNI.

The Java development team has provided many useful APIs. It can be a bumpy road integrating these APIs together, but SOA has proved it can be done successfully and efficiently.
Abstract

Science Opportunity Analyzer (SOA) assists scientists in planning a space mission through parameter computation and visualization. SOA is not your typical "run-of-the-mill" Java application since it requires a language that provides powerful APIs and must be multiplatform as well. SOA would also push the limit of some of these APIs. The APIs include used by SOA include: the Java Native Interface (JNI), Java3D, Java2D, Remote Method Invocation (RMI), and Reflection. This number and complexity of APIs in one package has provided SOA with several unique challenges that had to be resolved.

SOA combines many different APIs into one single application. JNI is used so that the Java portion of SOA can access a complicated navigation library, in C, that performs coordinate transformations written by experts. In visualization, the decision to use the Java APIs for graphics has allowed SOA to take advantage of Java3D's scenegraph structure and Java2D's simple yet powerful API. Reflection plays an important role in SOA in that it allows different Space Missions to add their specific components to SOA "on-the-fly" without having to recompile the application. RMI is used in SOA to allow a client to access a complex, third party mathematical software application. In general, these APIs give the programmers the ease of developing in Java and not having to worry about different standards for different languages on different platforms.

Integrating the various APIs has not been as simple as it sounds. Error trapping techniques for using JNI have created various issues including reporting errors. Using Java3D and Java2D also has some drawbacks including memory leaks and slow performance in early releases. Another major problem stems from having to visualize many objects that are very far away. Reflection was a new concept to the programmers at the start of the development phase requiring time to fully understand the API. Getting these APIs together made the software more complex. The difficulty of tracing null pointers, memory leaks, and plain old bugs in using the APIs proved overwhelming at times. There were countless sleepless nights that, in the end, resulted in finding ways to overcome these obstacles.

Some tools such as, Sun’s jvmstat, freeware JSwat, Rational’s OptimizeIt, and Quest Software’s JProbe suite have been used with varying results. Though they provide assistance to many problems, they are also lacking in debugging some of the APIs including Java3D and JNI.

The Java development team has provided many useful APIs. It can be a bumpy road integrating these APIs together, but SOA has proved it can be done successfully and efficiently.