On November 5, a peer review was held addressing the issue of Java use for flight implementations. This document summarizes the results of the review.

Summary of Conclusions

The panel was in general agreement that Java provides a number of significant advantages over conventional (Ada, C and C++) flight software implementation languages and that the presenters understood the technical risks associated with Java and had a reasonable strategy for introducing this language technology into JPL practice. All reviewers, except one, would recommend that the Mission Data System Project proceed cautiously with implementations of low risk modules requiring no real-time features.

Background

The choice of programming language for software applications has far-reaching implications related to programmer productivity, system reliability, system performance, and the ability to support a given software architectural model. Within the past 10-15 years, Object-Oriented (OO) designs, OO languages, and OO distributed system services have evolved as the industry best-practice for developing complex distributed systems, and is gaining acceptance as a best-practice in embedded real-time systems as well. Flight software and mission-critical ground software, developed with conventional procedural paradigms have been developed in Ada, and C, with real-time extensions to support time-deterministic behaviors. The addition of OO constructs to C resulted in the C++ programming language. However, in making these additions, C++ retained much of the idiomatic nature and weak typing which leads to poor software productivity and software defects. C++ allows OO programming but does not enforce it.

Recognizing the limitations of C++, and incorporating the latest thinking and research in language design, Sun Microsystems set about, in the early 1990s, to develop the Java language. This new language would both support and enforce the OO model and would also be capable of running on a variety of processor architectures without modification. These two characteristics are key to software reusability, which is a high leverage factor in software productivity and, in turn, our ability to build and launch missions faster and cheaper. The Java technology has now matured to a level where it is reasonable to use it to develop large applications. Given the broad mandate of MDS to maximize reuse and to
incorporate industry best-practice into the mission-critical flight and ground software at JPL, it is only fitting that MDS take the lead in attempting to exploit this capability.

Motivation for the Review

After a period of evaluation, the JPL Mission Data System (MDS) Project is currently planning to use Java in some of its non-real time, flight applications. Java is felt to have a number of key characteristics that ultimately will make it very attractive for mission-critical use and, in fact, is now being used successfully for mission-critical ground applications at JPL and GSFC. Because of the relative newness of Java, and the fact that there are no known spacecraft applications of Java in the aerospace industry, the adoption of Java is felt to carry with it a certain amount of risk, as compared to alternative approaches. In addition, a decision by MDS to proceed with Java for flight applications would establish a precedent. Other flight projects might make a decision to adopt this language technology based on that precedent, without a full understanding of the potential risks. Therefore, the primary motivations for this review were to: 1) review the material upon which the MDS Project made their decision; and 2) establish a precedent of sorts at JPL for the process of assessing and approving the adoption of a new programming language for critical applications.

Disclaimers

There was no intent at this review to assess the project implementation issues (cost and schedule) associated with the use of Java nor was there an attempt to assess the wisdom of the decision to adopt an Object-oriented paradigm. The review focused primarily on the identification and mitigation of technical risks associated with Java. Since it was understood that Java could not satisfy all MDS requirements, it was assumed that the use of Java would necessitate a mixed-language environment.

Note: there are no known NASA or JPL policies or procedures governing the process by which a new programming language is adopted.

Review Panel Members

The review panel was constituted primarily by individuals experienced either in Java programming or in real-time embedded software systems for spaceflight applications. Since there is no currently recognized standard Java real-time implementation, it was difficult (impossible?) to find an individual with real-time Java implementation experience.

We were fortunate to be able to have two individuals on the panel from outside JPL. Klaus Havelund is from the NASA Ames Research Center and is investigating technologies for Java verification. Nedim Fresko is from the Java language group at Sun
Microsystems, the company which developed Java and maintains the Java standards. The panel members and their affiliations were:

- Paul Backes 345
- Daniel Crichton 389
- Nedim Fresko Sun Microsystems
- Mike Girard 388
- Klaus Havelund Ames Research Center
- John Kelly 506
- Issa Nesnas 345
- Dave Nichols 738

Chairman

**Review Objectives**

The stated objectives of the review were to:

- Assess the technical risk evaluation done by the Java evaluation team.
- Assess the strategy being taken by MDS to understand and mitigate risk prior to wholesale adoption; and,
- Determine the extent to which significant additional risks may be present and recommend a strategy for addressing the risks.

**Summary of Answers to Questions**

The review panel was asked to answer five questions at the end of the review. The questions, along with a summary of the answers, follows.

1) *Do you agree with the arguments espousing Java's positive attributes?*

The panel all agreed that Java has a very compelling set of attributes that warrant its serious consideration for flight software. These attributes are:

- a) Significant reduction in development time vs. C or C++
- b) Significant reduction in software defects vs. C or C++
- c) Dynamic class loading and the Java compilation model can greatly simplify software updates during a mission.
- d) Direct support for the OO paradigm.
- e) A high level of portability from one Instruction Set Architecture (processor family) to another.
f) Improved ability to recruit and retain software engineering personnel at both the college grad and journeyman level.
g) Enhanced opportunities and lowered costs for re-use.

However, some of the advantages can be compromised when Java is used in a mixed-language environment. For example, interactions between C++ and Java memory management models could exhibit behaviors that would be difficult to debug with current tools.

2) *Have the presenters demonstrated that they understand all of the major risks associated with the use of Java for flight software?*

While it is unlikely that all risks have been characterized, the panel was in agreement that the presenters did an excellent job demonstrating that they understood the major risks that could be understood at this time. These major risks can be characterized as:

a) Multi-language development environment

The use of the Java Native Interface (JNI) as a mechanism to interface Java with code generated in other programming languages can pose memory management risks, particularly in association with garbage collection schemes. Robust debugging tools that directly support a C++/Java/JNI language environment do not exist. Multi-language debugging requires that programmers be fluent in multiple languages and interfaces. A multi-language environment also implies the necessity to integrate and maintain more vendor-supplied components.

b) Real-time

The Real-time Java specification has not been finalized. Extensive use of Java for flight applications depends on a robust Real-time specification and complete adoption by various vendors. Also, it is likely that Real-time implementations may not fully support the Java compilation model, possibly negating some of Java's positive attributes.

c) Performance

Initial experience with Java shows that it can be much slower than C or C++. This is certainly the case when compiled C/C++ code is compared to interpreted Java (which is the "normal" compilation model). However, Java is supported by other compilation models which can result in much improved performance, often comparable to C/C++. The ability to use these alternative compilation models in a time-critical, mixed-language environment has not been thoroughly demonstrated. It was also pointed out that Java does not support the numerical precision that may be necessary for some applications.
d) Maturity

Java is a relatively new language. As a result, debugging tools, compiler designs and garbage collection techniques have yet to reach the level of maturity that will ultimately be achieved and may at this point be somewhat buggy themselves. Because of the huge industry commitment being made to Java and the fact that Java is now the language of choice for study in academic computer science departments, this is expected to be only a short-term risk.

3) Has the analysis and evaluation addressed the major concerns?

The nature of this question has to do with whether or not the analysis performed and described in the evaluation report was complete enough to fully characterize the identified risk.

There was a sense among the panel members that the performance issues had not been adequately characterized to really understand the risk. In particular, it was suggested by more than one panel member that the performance assessment should be extended to include a comparison of data-structure heavy programs and OO features of the language (e.g. complex hierarchies, polymorphism). In addition, the benchmarks should reflect different styles of programming (e.g. floating point, allocation-intensive, networking, and I/O intensive).

Other areas that were identified as needing further assessment included:

- Debugging of multi-language applications
- Use of Rhapsody/UML (particularly automatically-generated code) in a multi-language environment
- Potential concurrency errors (deadlocks, thread starvation) in the Java multi-threading model and the JVM scheduler.

4) Where, in your opinion, is MDS most likely to run into problems with the use of Java?

The answers to this question generally parallel the identified risk areas. It was felt that if MDS were to adopt Java at this point in time (for non-real time applications), the problems they would most likely face would be related to:

a) Lack of performance
b) Difficulty in debugging
c) Assembling a development team with adequate fluency in Java

5) Do you see any reason why MDS should NOT proceed with their plans to use Java?

All panel members, except one, felt that the risks identified should not deter the MDS Project from proceeding with their plans to implement some carefully selected low-risk components in Java. Associated with this approach, however, was the suggestion that the Project: 1) undertake an effort to establish, validate, and assess the performance of the multi-language development environment; and 2) develop contingency plans in case the Java implementation runs into difficulties that would have an adverse effect on schedule.

The panel member who recommends against the use of Java (as compared to C++) expressed concerns about: 1) performance; 2) language power and flexibility; 3) mathematical expressions; 4) difficulties in implementing certain constructs; and 5) lack of maturity.

**Summary of Recommendations**

More work is needed on characterizing performance.

Assess the impacts of a mixed-language implementation on the Rhapsody modeling approach.

Provide parallel implementations in C++ for any early Java implementations. Besides providing a backup, this will allow a direct comparison of the two approaches.

Collect and publish (on the web) the lessons-learned as this pioneering activity proceeds.

Establish a formal, recurring reporting activity (to the stakeholder projects) on the key risk areas, particularly performance.

An initial Java team of one person is inadequate. Critical mass is at least three or four people. This is necessary for the team to develop a fluency in key areas such as testing, concurrency, determinism, memory management, etc.

Initial efforts should require minimal or confined use of JNI.

Aggressively participate in the public response to the Real Time Specification for Java.

MDS should look for ways to do dynamic class loading as hardware capabilities increase.

Before proceeding, MDS (or any Project considering the use of Java) should identify candidate modules for Java implementation and review the decision relative to the risks.
MDS should proceed as soon as possible to develop and validate a mixed-language development environment.
## Participants

<table>
<thead>
<tr>
<th>Name</th>
<th>Email</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dave Nichols</td>
<td><a href="mailto:David.A.Nichols@jpl.nasa.gov">David.A.Nichols@jpl.nasa.gov</a></td>
</tr>
<tr>
<td>Paul Backes</td>
<td><a href="mailto:Paul.G.Backes@jpl.nasa.gov">Paul.G.Backes@jpl.nasa.gov</a></td>
</tr>
<tr>
<td>Kirk Reinholtz</td>
<td><a href="mailto:Kirk.Reinholtz@jpl.nasa.gov">Kirk.Reinholtz@jpl.nasa.gov</a></td>
</tr>
<tr>
<td>Dan Crichton</td>
<td><a href="mailto:Dan.Crichton@jpl.nasa.gov">Dan.Crichton@jpl.nasa.gov</a></td>
</tr>
<tr>
<td>Issa Nesnas</td>
<td><a href="mailto:Issa.A.Nesnas@jpl.nasa.gov">Issa.A.Nesnas@jpl.nasa.gov</a></td>
</tr>
<tr>
<td>Mike Girard</td>
<td><a href="mailto:Michael.A.Girard@jpl.nasa.gov">Michael.A.Girard@jpl.nasa.gov</a></td>
</tr>
<tr>
<td>John Kelly</td>
<td><a href="mailto:John.C.Kelly@jpl.nasa.gov">John.C.Kelly@jpl.nasa.gov</a></td>
</tr>
<tr>
<td>Rick Grammier</td>
<td><a href="mailto:Richard.S.Grammier@jpl.nasa.gov">Richard.S.Grammier@jpl.nasa.gov</a></td>
</tr>
<tr>
<td>Frank Kuykendall</td>
<td><a href="mailto:Frank.Kuykendall@jpl.nasa.gov">Frank.Kuykendall@jpl.nasa.gov</a></td>
</tr>
<tr>
<td>Wallace Tai</td>
<td><a href="mailto:Wallace.S.Tai@jpl.nasa.gov">Wallace.S.Tai@jpl.nasa.gov</a></td>
</tr>
<tr>
<td>Al Sacks</td>
<td><a href="mailto:Allan.L.Sacks@jpl.nasa.gov">Allan.L.Sacks@jpl.nasa.gov</a></td>
</tr>
<tr>
<td>Dan Erickson</td>
<td><a href="mailto:Daniel.E.Erickson@jpl.nasa.gov">Daniel.E.Erickson@jpl.nasa.gov</a></td>
</tr>
<tr>
<td>Glenn Reeves</td>
<td><a href="mailto:Glenn.E.Reeves@jpl.nasa.gov">Glenn.E.Reeves@jpl.nasa.gov</a></td>
</tr>
<tr>
<td>Klaus Havelund</td>
<td><a href="mailto:Havelund@ptolemy.arc.nasa.gov">Havelund@ptolemy.arc.nasa.gov</a></td>
</tr>
<tr>
<td>John Lai</td>
<td><a href="mailto:Jlai@jpl.nasa.gov">Jlai@jpl.nasa.gov</a></td>
</tr>
<tr>
<td>Abdullah Aljabri</td>
<td><a href="mailto:Abdullah.s.aljabri@jpl.nasa.gov">Abdullah.s.aljabri@jpl.nasa.gov</a></td>
</tr>
<tr>
<td>Anne Elson</td>
<td><a href="mailto:Anne.Elson@jpl.nasa.gov">Anne.Elson@jpl.nasa.gov</a></td>
</tr>
<tr>
<td>Chris Jones</td>
<td><a href="mailto:Chris.P.Jones@jpl.nasa.gov">Chris.P.Jones@jpl.nasa.gov</a></td>
</tr>
<tr>
<td>Peter Gluck</td>
<td><a href="mailto:Peter.R.Gluck@jpl.nasa.gov">Peter.R.Gluck@jpl.nasa.gov</a></td>
</tr>
<tr>
<td>Dave Eisenman</td>
<td><a href="mailto:David.J.Eisenman@jpl.nasa.gov">David.J.Eisenman@jpl.nasa.gov</a></td>
</tr>
<tr>
<td>Vicki Shipkowitz</td>
<td><a href="mailto:Vicki.Shipkowitz@Eng.Sun.COM">Vicki.Shipkowitz@Eng.Sun.COM</a></td>
</tr>
<tr>
<td>Nedim Fresko</td>
<td><a href="mailto:Nedim.Fresko@Eng.Sun.COM">Nedim.Fresko@Eng.Sun.COM</a></td>
</tr>
</tbody>
</table>