

RNJ Review Summary

Jim Kowalkowski, Marc Paterno

1 Introduction

This document is a summary of our thoughts concerning the review of the ROOT and Java (RNJ) project proposed by Philippe and Suzanne. The information here is mostly a record of what we already discussed during our meeting on January 19; we have added a small amount of additional commentary.

2 Scope of Project

Our understanding is that there are really two phases to this effort. To summarize:

- *The first phase provides a low-level interface to ROOT that will enable it to forward all drawing command to a remote Java application. It also provides a message protocol that can be used to transfer and dispatch the byte streams put out by ROOT object streams.*
- *The second phase provides interaction with higher-level ROOT objects such as Histograms (TH1D) within Java*

In phase 1, the Java application interacts with ROOT using low-level events such as “right-mouse-button-press” and ROOT sends low-level primitive drawing commands to the Java application. The primary immediate customer is CDF online and the primary use case is to bring up histograms from home or home institutions. The project is not necessarily scoped to complete the second phase on a timetable appropriate for Run II.

3 General Comments

The manpower request (2 people, 5 months for phase I) represents a considerable investment of the time of two talented people, Philippe and Suzanne. This is effort devoted directly to providing an additional feature to ROOT — code that is not useful outside of ROOT. We are not sure that taking this approach is in the best interest of the PAT group or its customers. Sections in this document explain other approaches that could be explored, most of which are things we discussed during the January 19 meeting. This concern is not due to the project itself or the ideas within it; rather it is a concern about the costs versus the benefits of this approach. If the scope of phase I was such that it could be done quickly (1-2 months, with two people), then we would not have this concern.

The initial list of users for this package (CDF online) seems very small, and the primary use case is also limited. One thing that worries us is that if RNJ phase I does not live up to expectations, or if it is delayed, that it will not be used. Basically, we are concerned that if the performance isn't clearly better than what one would achieve using X, the product will not be accepted by the users; if the features are limited, or if the ease of use is degraded with respect to X, the same will result. We suggest more users be consulted to ensure that there is a definite user base for this product at Fermilab, that the feature list is correct, and that it will not fall short of expectations — that users understand exactly what they are getting. In particular, Jae Yu, of the DØ online monitoring group, should be consulted as soon as possible. When we first discussed this with Jae, he quickly indicated that he was interested. When we went into detail of exactly what would be developed, he was not convinced that it would be adequate.

The fact that this code must be kept in sync at all times with ROOT brings up the question of maintenance. The ROOT team changes ROOT internals regularly; this may pose a problem. We are concerned that the continuous maintenance burden, lasting beyond the five month timescale, may be unacceptable for the PAT group to support.

We believe that users really want to produce applications that interact with ROOT objects from Java. In other words, they really want phase 2 now. We are concerned that a significant fraction of the development of phase I does not address this need, and suggest consideration of a slightly different initial direction, which more immediately supports the goals of phase 2. We have not talked with many of the targeted users of this package, so we could be mistaken.

4 Specific Concerns

4.1 X Server

We are concerned that the ROOT “VirtualX” implementation may present problems. The alternative of producing the six or so other classes (TCanvas, etc.) presents the same problems. The “VirtualX” implementation is likely to suffer from the same trouble that X does — lots of network traffic, which means high bandwidth requirements. We do recall that it was stated that higher-level objects can be recognized on the Java side if performance is an issue. This means more development effort, and we have no good estimate of how much additional effort is required. It also means tighter coupling to specific ROOT objects. This is the crux of our most significant concern: if the performance of RNJ isn’t significantly faster than X, with the same reliability as X, and with the full set of features of the X solution, then the product will not see wide use. We think it is important to have a shorter-term goal of a useful product, that can be assessed on the timescale of ~2 months.

We also wonder about the “look and feel” of the ROOT GUI in Java. Experienced ROOT users will be accustomed to a certain style of display (menus, dialogs, etc.) and we don’t understand how much effort will be required to reproduce this style in Java. These details are notoriously painful to deal with, but are important if the user interface is to be usable on several platforms. These style issues will grow worse if there is a performance problem requiring more ROOT classes to be implemented in the Java world.

4.2 Architecture Style

We are concerned with the synchronous nature of interactions with the server and the strong one-to-one relationship between a ROOT object on the client and one on the server. This distributed architecture appears to be mostly geared towards files or blocks of shared memory on the server. This may be too restrictive for some potential users. This is not the general architecture of DØ, for example, where many users of the online monitoring can connect to an online process and request notification of histograms. These histograms can be manipulated in the client independent of the server process (panning, labeling, etc). The “rendering” engine puts a large burden of GUI manipulations and state into the server. Since the ROOT objects hold GUI state and data for the thing they represent, multiple users can perhaps interfere with each other or cause much replication of the same data. This is a design flaw in ROOT that will have to be dealt with by RNJ.

4.3 Reproducing Existing Functionality

Existing packages that use CORBA or Java RMI already have distributed architectures and protocols. We are afraid that this package is reinventing these facilities. Since we discussed this extensively in the meeting on January 19, we’ll not belabor it here.

4.4 Streamer Versioning

We do not understand how the streamers will be kept in sync from the Java client to the ROOT server. How will objects in older files be interpreted? The ROOT model for streamer versioning may be difficult to maintain and duplicate in the Java world.

5 Specific Recommendations

In this section, we summarize (or just recall) some specific recommendations we discussed during the January 19 meeting. These suggestions do not necessarily form a complete whole; each addresses a different topic.

5.1 *Threads*

We strongly recommend looking into using threads on the server side to control and operator the communications protocol and interact with the ROOT engine.

5.2 *XML as Interchange format*

As a general point, we suggest making use of existing standards whenever possible. An example would be doing research on standardizing on an XML format for a histogram with Tony Johnson (JAS), and establishing that as the standard exchange format for ROOT histograms. If such a standard format were available, a simple object living on the ROOT server side could convert a TH1D (for example) into XML data and ship it to tools that can understand and display XML data. If the primary user of this system wants to display histograms, the JAS tools or other XML capable tools may provide more than adequate plots and graphs. A tool like this could open the ROOT engine to be used with other tools and also reduce the amount ROOT specific code that the PAT group will need to support and maintain.

5.3 *Use of CINT and Java*

Java has a reasonably complete meta-object protocol, through which classes and objects can be introduced at run time. It is conceivable that the ROOT CINT run-time dictionary can be used to generate Java classes on the fly as the program runs. The classes produced on the Java side would model what a user sees at the CINT prompt, not necessarily the exact class hierarchy seen in C++ code. In other words, if a subclass is derived from two base classes, then the Java class could be the combination of all three C++ classes. The interactions with objects living in the ROOT server from the Java client would be through an interface; the object is not transferred to the client by value, it really lives on the server and the object interface is extended to the client. This pattern is consistent with CORBA, Java RMI, and DCOM, which means that these distributed components can conceivably be used to handle the network messages transfers, coordination, and dispatching. Managing network software components have been traditionally a severe maintenance burden. Using a standardized protocol (we suggest CORBA or RMI) also allows the future expansion and the possibility of using different front-end technologies.

Generating classes on the fly poses the problem that the meta-object facility must be used to call the methods of the class. It may be difficult to design a flexible system that is also easy to use. An alternative is to produce a program that generates CORBA IDL definitions or Java classes from the information in the CINT dictionary, and use that to build the Java applications.

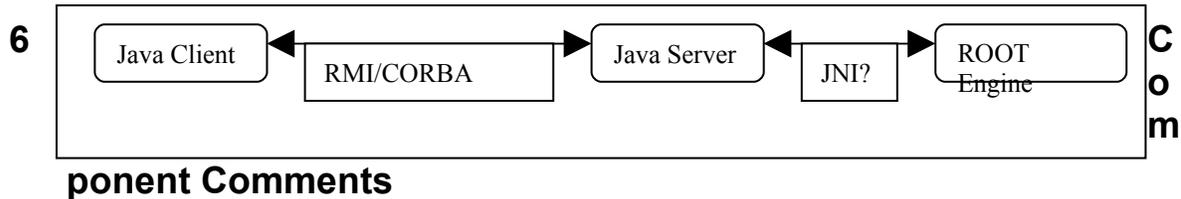
Another problem with generating classes is the use of arrays and other container classes. Many times container constructs are completely incompatible between languages. Since ROOT contains fairly simple container classes, we expect they can be interfaced to Java without unreasonable effort.

The CINT dictionary contains all the information needed to browse objects from the CINT prompt. These facilities might be preferred over the general streamers because they can always provide constant data on demand from an object. A Java client could ask for the object definition from the CINT dictionary (not using the streamers) and then request the object's data.

5.4 *Alternative Architecture*

We suggest the consideration of an architecture that pushes the Java virtual machine in the ROOT server process. This implies that Java must run reliably on the server machine. This may be Linux, which is going to be the client architecture for many users – so Java must be supported there anyway. It is important to note that the Java that runs in the server will be used to serve objects to Java clients – not only to run Java

GUI applications. Many of the Java virtual machines run great as servers where no GUI interactions are required.



The key components to phase 1 will be used to develop phase 2. These key components needed for phase are included in this section.

6.1 Threads

We suggest that the client model for using threads in the prototype code and described in the review will not be adequate. If threads are going to be used, then the network components need to operate independently of the processing or GUI interactions.

6.2 Protocol Specifications

If the system does not rely on an established standard such as CORBA or RMI, then a protocol to transfer ROOT object byte streams reliably between C++/ROOT and Java is needed. This was inadequately specified. A protocol specification should including all the protocol messages and transactions that can occur. A state diagram that documents what happens during errors, disconnect, reconnect, and normal transfers should be completed before any coding happens. Development of such a protocol is, of course, a significant effort, which is one of the reasons why we suggest the user of CORBA or RMI, which handles this communication.

6.3 Object Dispatching

We believe that the object dispatching on the Java side is not scalable. In a multithreaded environment, the objects should be dispatched to the listeners. Using an enumeration for each class can only lead to poor code constructs (large case statements). It would be preferable for the listeners to get notified when as object of the type they want appears. There are many discussions and examples of event dispatching in *C++ Report* and *JOOP*.

6.4 Garbage Collectors

We don't understand the comment in the proposal about prodding the garage collector every so often. The point of garage collectors it so users do not need to be concerned with such details – the collector does its work when it needs to. Unless some profiling demonstrates that a JVM's garbage collection is inadequate, we suspect such action may be counter-productive.

7 Proposal

We were asked to give our opinion on what we feel the organization and goal of this project should be. We realize that this is a difficult thing to do because of our distance from the project and limited knowledge. This list should not cause you to ignore other items in this document because they are not included here.

1. Evaluate the possibly of creating a “javaRootCINT” utility for Java that acts like the rootcint program. This is the program that produces the Java classes from C++ headers (couple of weeks effort)
2. Evaluate the possibly of using Java in the ROOT server end. You may know the answers here already – is there good JVM available for the platforms we need to use.

3. Evaluate the possibility of using a CORBA model for extending the ROOT object interfaces to the client. This should mostly involve discussions with people from the lab that have experience working with these tools (D0 data handling and perhaps CDF DAQ & controls).
4. If a custom protocol must be used to transfer data, then design it and design a real dispatching component on the client side. Test a prototype of this thoroughly with a couple different object types. This includes other suggestions in this document such as fixing the multithreading.
5. Do not spend a lot of time on the java-root console window. Any simple terminal emulator can suffice here; time will be better served prototyping other parts of the system. In addition, do not spend a lot of time getting the Java clients running in the web browser until more is understood about the direction of this project.

This is as far as we would like to go with this list. After many of these things are complete, the next steps are either obvious (development of the “VirtualX” class) or depend on the results of 1-4. Like we write at the beginning of this section, this list is hard for us to determine. We would be happy to discuss this in person.

8 Conclusion

There is really no detail in this report due to lack of time. We are more than happy to discuss this project further in person. The major concern here is the time it will take to create a ROOT specific add-on that may have very limited appeal. Again, time for experienced developers such as Philippe and Suzanne is very valuable. Will Philippe be able to work on this package for 5 months full time? The deadline for this project could slip drastically if Philippe cannot focus on this project directly. It may not be possible for this focus to happen. Has the time estimate taken existing responsibilities into account? Does the deadline match the 5 month, 2 person manpower estimate (deadline of July 1, 2000)?