This interface was initially conceived almost 2 years ago. The initial implementation was available within a few weeks. Since then the ideas have matured and evolved to provide a unique reference mechanism which we feel makes it an ideal mechanism by which inter-system interfaces might be constructed. The approach relies on reflectance in the underlying languages being interfaced.

1 Other Inter-System Interfaces

There are a few other approaches to similar inter-language interfaces. Some have different characteristics and can be considered as alternatives. Others are similar in goals.

S-Plus 6 The S-Plus6 Java interface is unfortunately a subset of this interface. However, if there would be little effort in adopting this implementation to provide their functionality within S-Plus and additionally references, non-static methods and fields, dynamic compilation, callbacks to S functions, mutable objects, etc. Additionally, the approach of expressing callback "functions" as strings causes major problems. Firstly, there is not facility for referring to the same instance of an object or function. One is forced to use global variables which causes major difficulties in a threaded world. Secondly, debugging is made complicated as there is no particular semantic information passed in the callback object, but instead it is parsed at a later time. Additionally, consider the idea of substituting values into the callback. This is made complicated if not impossible. And lastly, the entire concept of re-entrant calls from different threads to the single S-Plus evaluator seems to have been overlooked.

CORBA The dynamic CORBA package in R and S allows an R/S session invoke methods in one or more Java applications. This mechanism has the added advantage of allowing inter-process and inter-machine communication. This embedded R-Java interface takes advantage of the single process approach so that communication can be done on objects within the same address space.

XML Static or off-line exchange of data can be done by representing the data in XML format. The structured, self-describing nature of XML allows it to be application (and platform) neutral meaning that a single object expressed in XML can be potentially read by many applications with minimal exporting, conversion and basic management by the user between all pairs of applications. This is a different form of dynamic or "live" communication between systems, but does offer persistence.

The XML library for both R and S, and the abundance of XML parsers for Java, *C* and *C++* make it relatively easy to process XML input. Exactly how it is interpreted is left to the application, but the general structure is supplied in the XML file itself.

Streams Low-level inter-system communication can be done with a well-define protocol agreed on by the two parties involved in the exchange. The mechanism by which the data are transmitted can be basic streams: files, FIFOs, sockets, etc. This approach is to be avoided if possible since it places an unnecessary burden on the developer; requires multiple implementations of the sending and receiving routines that process the data being communicated; provided little in the way of extensibility as the communication and packing mechanism are data-type-specific; and, finally, is unfriendly in debugging at the user-level. This approach should be familiar to users of PVM – the Parallel Virtual Machine – and MPI – the Message Passing Interface – popular in scientific computing.

This approach has been the most common over the past 10 years. Some have used very low-level communication channels such as FIFOs and sockets. Others have used RPC – Remote Procedure Calls – which somewhat hides the packing of the data.

2 Features of the R-Java Interface

References The approach used in the R-Java interface is unique, but obvious. The concept of foreign references is not very innovative, but it is powerful and, to our knowledge, has not been used in this type of communication. The concept is simple: objects created in one language are operated on in that same language. The object can be exported to another language as a reference, allowing it to be operated

on by the other language, but by functions in its own language. Pass-by-reference systems such as this are more general than pass-by-value *only* systems. This is because one can always extract the state of an object by recursively extracting its primitive components. For example, a matrix passed as a reference can be manipulated in Java with side-effects being communicated to R. However, the Java classes can elect to extract the data from the matrix and operate on them independently of R via a `getElements()` method. (This would be implemented via an R function such as *as.numeric()*.)

The mutable-state of the references is a vital aspect of this interface. It allows the Java methods to modify the value of an R object in a well-defined and traceable manner. The S4 version provides a similar facility without the benefit of closures.

Dynamic Compilation The ability to dynamically define and load Java classes which implement a particular Java interface makes it easy to use newly encountered classes. For example, suppose we use the dynamic classpath mechanism of Omegahat to introduce classes to a running Omegahat session. To allow an R function to implement an interface and then use such an object in a computation, we can simply compile a new Java class. This avoids having to manually create a new class, restart the session and obtain the same state.

Omegahat In addition to providing the basic method dispatching used in the R-Java interface, the Omegahat interpreter provides many useful facilities for an interactive session. These include dynamic class management and compilation; variable and database management; functions; partially qualified class names; dynamic CORBA method invocation and servers; system monitoring tools (classpath viewer, class inspector, toggleable, per-method profiling tools, ...). Additionally, statistical data structures and tools such as optimization, genetic algorithms, distributed computing (via CORBA), distribution estimation and random generators provide a rich set of facilities on which to build further Java classes.

Converters The heart of an inter-system mechanism is the ability to express how values are to be translated between the systems. The dynamic, user-extensible and user-controllable approach to specifying how objects are converted in both directions makes this interface easy to modify in "real-time". Converters can be implemented as C routines or R functions, making it efficient and convenient.

3 Future Enhancements

$ accessors It would be convenient to mimic the syntax we provided in S4 for CORBA that uses the $ operator as an accessor on a foreign object. For example, given a variable *jobj()* containing a reference to a Java object, we could invoke a method `plot()` as

```
jobj$plot(x, y)
```

This even allows the arguments to have names for the Java databases

```
jobj$plot(x, j1=y)
```

which would store the converted object for *y()* in the named Omegahat database under the name `j1`.

Multi-dimensional Array Conversion Currently, we have disabled the automatic conversion of multi-dimensional arrays. (This is done in the Java side so they are not even seen by the C code.) We can decide how to deal with these types of arrays: converting them to lists of lists ... of primitives or to multi-dimensional arrays. The former allows us to handle ragged arrays, but the latter may be more useful.

Graphics Device The Java graphics facilities (Swing, 2D and 3D) provide a powerful environment for creating graphical displays with many features. We can implement a Java graphics device by implementing the necessary C-level calls via JNI to pass the drawing instructions to a Java component. This style of device is nice as it allows the component to be used in arbitrary locations within different GUIs. For example, the device might be a button or a single component in a window.

Garbage Collection It is possible to determine from within an R call to *.Java()* or *.JavaConstructor()* whether an anonymous reference is assigned to an R variable. If not, the object can be release automatically by the R side at the end of the call. For example, in a call such as

```
.Java( .Java(o, "getClass"), "getName")
```

the result of the inner *.Java()* call can be removed from the anonymous Omegahat database.

Threads We need a more structured mechanism for handling threads. This requires changes to the R engine and we are hesitant to upset the other developers there by proposing somewhat major changes. Some of the developers (Luke , Ross and Rob) have been looking at micro-threads. How these interact with Java threads remains to be seen. A minimum requirement is that we ensure that functions called from Java threads as callbacks to R foreign references do not get executed in a separate thread while the R evaluator is performing another task.