Approaches to R as web and analytic service

Nianhua Li, Martin T. Morgan, Seth Falcon, Robert Gentleman

Fred Hutchinson Cancer Research Center
1100 Fairview Ave. N.
PO Box 19024 Seattle, WA 98109

15 Sept, 2006

Abstract

The aim of *RWebServices* is to facilitate exposing R functions as effective Java-based web or grid services. The application uses existing technologies in web and grid services, XML, as well as new and existing inter-system interfaces between R and other languages. We briefly review the state of art of these research areas here, with an emphasis on applications related to *RWebServices*.

1 Introduction

Web services [1, 2] refer to a set of standards and techniques that allow applications to be described, published, discovered and invoked over an internet protocol backbone. Web services enable applications to communicate in a platform and language independent way, increasing software sharing and reuse. Multiple programming languages have been used in web services, such as Java [3, 4], Perl [5], C++, C#, Python, PHP, VB, and Ruby.

A *grid service* is a web service specializing on a computing environment that integrates distributed and heterogeneous resources across various administrative organizations [7, 8]. An important distinction between a grid service and a general web service is the support of stateful resources, i.e., services can be created and destroyed during run-time. The high-level definitions of grid service architecture and its requirements are given by the Open Grid Services Architecture (OGSA) [6, 9]. The Web Services Resource Framework (WSRF) [10] provides detailed specifications for stateful resource management using web services technology. There are implementations of OGSA and WSRF in various languages. Example implementations include the Globus Toolkit in Java and C++ [11], WSRF::Lite in Perl [12], WSRF.NET in Microsoft .NET framework [13], and pyGlobus in Python [14].

Web and grid service support for R is still in its early stages. An effective way to connect R with web or grid services is to leverage existing efforts in

other language domains, in particular using Java as a conduit to provide R web and grid services. We chose Java rather than other languages because web and grid services support is well-established in the Java community, and because communication with third-party Java components is a feature of the *caBIG* project.

This document discusses systems integrating R and Java, and provides a brief review of existing web and grid services support in the R community. The document concludes with a brief synopsis of formats for statistical data exchange.

2 Bridging R and Java

Two different approaches have been used to achieve interoperability between R and Java: message streams and shared libraries.

2.1 The message stream approach

In this approach, R and Java are run as separate processes. Data is transported as message streams with an encoding that can be understood by both parties. Message streams can be text-based or binary. Transportation media can be socket connections, web services over the internet, or even file systems (standard I/O). Open Statistical Services (*OSS*) [15] and *Rserve* [16] are two examples of the message streams approach.

This approach is useful if the R component may potentially talk with other language systems in the future. New language systems can be easily included, since the only requirement is serialization and deserialization between the message encoding and language specific data structures. Moreover, it is known that R is not thread-safe, whereas Java supports multi-threading. Because R and Java processes are independent from each other in this approach, it is possible to have multiple Java threads communicating with R concurrently.

There are several disadvantages to a message streaming approach. First is efficiency: data is serialized and deserialized at every stage during transportation. Second, stream-based transportation is slower than direct programming calls, no matter what encoding or transportation mechanism is used. The third concern is related to data persistence and error handling. Because R and Java belong to separate processes, extra implementation is required to ensure the ability to deliver error messages even after after a server or data transportation failure. Security is also a challenge in server-client architectures. The final challenge is session management: imagine that a Java process invokes an R function which uses objects of the caller (the Java process). Both parties need server support, and the transportation layer needs to support session management.

Open Statistical Services (*OSS*) integrates Java and R via StatML, an XML-based markup language for encoding statistical data. *OSS* contains two major components: the *Java bridge* and the *R bridge*. Both components support data serializations between StatML and data structures in R (the R bridge) and Java

(the Java bridge). The message transportation channel can be either standard I/O, or internet-based web services. A message of R method invocation from Java includes two parts: the function name as a character string, and input parameters in StatML encoding. Basic R data types including numeric, integer, logical, and character vectors, as well as list and array, are covered by the StatML encoding. The StatML encoding is text-based for scalars, and 64-bit binary for lists and arrays. Invocation of Java methods from R is performed in a similar manner.

Rserve is a TCP/IP server that allows R to be invoked from other applications. Session management is supported. For example, a client can first store data on *Rserve*, then modify the R environment on *Rserve* through several method invocations, with intermediate results kept on the server, and finally retrieve results. Method invocation follows normal R syntax, and is sent as a series of text messages. Data is sent in binary forms using an in-house serialization protocol. The current implementation supports basic R data types including integer, numeric and character vectors. Complex data types such as S3 and S4 objects can also be encoded using a low-level interface, but the decoding part has not been implemented yet. The unique part of *Rserve* is that it forks a new R process for each client. Therefore multiple clients can be supported simultaneously. This cannot be done with standard R. *Rserve* also provides error handling and security support. There are C/C++ and Java clients available for *Rserve*.

2.2 The shared library approach

R and Java can also be integrated into a single process, with one embedded into the other. There is no direct programming interface between R and Java. However, the interface between Java and C is well-developed in the Java Native Interface (JNI), and the internals of R itself are largely written in C. The technique of bridging R and Java via C has been used by applications such as *SJava* [17], *arji* [18], and *JGR* [19] and related tools (*JRI* and *rJava*).

The shared library approach has two advantages. First, the interface (C) is a programming language, which is more powerful than any encoding mechanism. It therefore gives more flexibility to integrate languages. Secondly, both R and Java are in a single process, avoiding challenges in a typical client-server architecture such as session management, data persistence and security. Error handling is also easy inside one process, because both JNI and the R interface to C have error handling mechanism. Some applications also provide advanced features such as callback and object references. These concepts are easier to implement than when using the message stream approach.

There are also downsides to the shared library approach. Because R is not thread-safe, parallel processing requires additional layers even though Java supports multi-threading. Also, the learning curve for low-level application development is steep, because interactions between at least three languages are involved.

SJava can be used to embed `R` in `Java`, or to embed `Java` in `R`. Interesting features of *SJava* includes callbacks, object references, and customized data conversion. For example, the `R` instance embedded in `Java` can call back to the original `Java` virtual machine. Object references allow data stored in one language format to be referred to in another language. Thus a `Java` data structure (e.g. `Vector`) can contain a reference to `R` data. This avoids unnecessary between-language data transformation and evaluation. Data conversion in *SJava* is highly configurable yet easy to use. Built-in converters map `R` character, integer, numeric and logical vectors to `Java` primary type arrays or existing classes in `Java`. Users can provide additional converters written in `C` or `R` to extend or overwrite the default data transformation behavior. These convert functions can be "high-level", with details of the data conversion handled by *SJava*. Moreover, the registration of user-level converters in *SJava* is performed during run time via a well-defined interface. Therefore, the transformation of arbitrary or complex data types is achieved with minimal efforts.

JGR is an integrated development environment for `R` written in `Java`. Its `Java`-based graphical user interface communicates with `R` through two facilities: *JRI* (Java/R Interface) for `Java` to `R` and *rJava* for the reverse task. Both provide low-level `R-Java` interfaces using JNI. Yet another example of the shared library approach is *arji*, a Bioconductor package similar to *rJava*.

3 Bridging `R` and web or grid services

Web and grid services support in the `R` community is receiving increasing attention. At least two applications have been developed in the last year to transform `R` and `Bioconductor` packages into web services.

RProteomics uses Open Statistical Services to integrate `R` with `Java`. The `Java` application is further developed as a *caGrid* analytical service, which offers semantic interoperability and security control in addition to classical web service capabilities. *RProteomics* focuses on proteomics data analysis.

The Comprehensive R-based Microarray Analysis web service (*CARMAweb*) [20] contains a `J2EE` front end and an *Rserve* back end. It provides a web interface and a web service interface. For a typical service request in *CARMAweb*, the method being invoked and tuning parameters are specified as primary data types (e.g. character, integer). Input data sets are provided by files. The `J2EE` front end organizes input information into a LaTeX document, and transfers it to `R` via *Rserve*. Data analysis in R is wrapped by *Sweave*, so that *Sweave* extracts `R` commands from LaTeX and packs results back to a document (LaTeX, converted to PDF). Results are returned as PDF files, and targeted for human interpretation. This is in contrast to typical web services where service responses are expected to be interpreted by other applications.

Finally, it is worth mentioning efforts to expose `R` functionality through a web interface. This approach allows end users to take advantage of the statistical computing power of `R` without tedious command line programming. `Perl` is often used as a bridge between `R` and the web. Examples include *MIDAW* [21]

(MIcroarray Data Analysis Web tool), *RACE* [22] (Remote Analysis Computation for gene Expression data), and *webbioc* [23].

3.1 Statistical data exchange formats

Statistical computations use specialized concepts (e.g., `NA`, corresponding to 'missing' data observations) that do not exist in general-purpose programming languages. Bridging `R` with other web or grid services requires either that these data structures be described in a way that facilitates data exchange, or the services exposed by `R` make use of general purpose data representations that allow interoperability without sacrificing too much expressivity.

There are three efforts that propose data exchange formats for statistical data: `StatDataML` [24], `CDFML` [25] and `StatML`. All of these formats are XML-based markup languages aiming to facilitate the exchange of statistical data across platforms. They have been used in communication between `R` and other systems such as `MATLAB` and `Java`. McConnell [15] provides a review of these data exchange formats.

There are several limitations to employing a formalized data exchange model such as those outlined in the previous paragraph. First, effective interoperability requires implementation of serialization mechanisms in each target language. Second, data exchange formats require integration into web or grid service standards like SOAP or WSDL. Third, no clear consensus on data exchange format is available, leaving little hope for interoperability amongst services implementing a single data model. Finally, from the perspective of communication between `R` and the web- or grid-services front end, the 'shared library' approach is much less dependent on formalized serialization models than is the 'message stream' approach.

The limitations sketched in the previous paragraph point toward use of light weight and *ad hoc* data structures that capture key features of the statistical domain, without requiring complex serialization mechanisms or data structures in non-statistical languages. This approach provides a short-term solution that enables exchange of statistical data amongst a diversity of programming languages, while sacrificing the expressivity of a carefully structured exchange format and the long-term benefits of interoperability such formats provide. This approach does not preclude the implementation of structured exchange formats once patterns of usage become well-established.

References

[1] D. Booth, H. Haas, F. McCabe, E. Newcommer, M. Champion, C. Ferris, and D. Orchard. Web Services Architecture. *W3C Working Group Note*, 2004.

[2] E. Cerami. Web Services Essentials, Distributed Applications with XML-RPC, SOAP, UDDI & WSDL. OŔeilly Media, 1st edition, 2002.

[3] Java Technology and Web Services. http://java.sun.com/webservices/

[4] R. Irani, S. Basha. AXIS: Next Generation Java SOAP. Peer Information, 1st edition, 2002.

[5] R. Ray and P. Kulchenko. Programming Web Services with Perl. OŔeilly Media, 1st edition, 2002.

[6] I. Foster, H. Kishimoto, A. Savva, D. Berry, A. Djaoui, A. Grimshaw, B. Horn, F. Maciel, F. Siebenlist, R. Subramaniam, J. Treadwell, J. Von Reich. The Open Grid Services Architecture, Version 1.0. http://forge.gridforum.org/projects/ogsa-wg

[7] I. Foster, C. Kesselman, S. Tuecke. The Anatomy of the Grid: Enabling Scalable Virtual Organizations. *International Journal of Supercomputer Applications*, 15(3):200-222, 2001.

[8] I. Foster, C. Kesselman, J. Nick, and S. Tuecke. The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration. *Globus Project*, 2002. http://www.globus.org/research/papers/ogsa.pdf

[9] J. Treadwell. Open Grid Services Architecture Glossary of Terms. *Global Grid Forum OGSA-WG*. GFD-I.044, 2005. http://www.ggf.org/documents/GWD-I-E/GFD-I.044.pdf

[10] K. Czajkowski, F. D. Ferguson, I. Foster, J. Frey, S. Graham, I. Sedukhin, D. Snelling, S. Tuecke, and W. Vambenepe. The WS-REsource Framework, Version 1.0. 2004. http://www.oasis-open.org/committes/download.php/6796/ws-wsrf.pdf

[11] I. Foster. Globus Tookkit Version 4: Software for Service-Oriented Systems. *IFIP International Conference on Network and Parallel Computing*, Springer-Verlag LNCS 3779:2-13, 2005.

[12] WSRF::Lite - An Implementation of the Web Services Resource Framework. http://www.sve.man.ac.uk/Research/AtoZ/ILCT

[13] M. Humphrey and G. Wasson. Architecture Foundations of WSRF.NET. *International Journal of Web Services Research*, 2(2):83-97, 2005.

[14] Python Globus (pyGlobus). http://dsd.lbl.gov/gtg/projects/pyGlobus/

[15] P. McConnell, R. Haney, S. Mungal, M. Peedin, and S. Lin. Open Statistical Services. caBIG Documentation, 2005.

[16] S. Urbanek.Rserve-A fast Way to Provide R Functionality to Applications. In *Proc. of the 3rd International Workshop on Distributed Statistical Computing (DSC 2003)*, 2003. ISSN 1609-395X, Eds.: K. Hornik, F. Leisch, and A. Zeileis.

[17] D. Temple Lang. The Omegahat environment: New possibilities for statistical computing. *JCGS*, 9(3), 2000.

[18] V. Carey. arji - another R-Java interface. http://www.bioconductor.org/packages/1.9/bioc/html/arji.html

[19] M. Helbig, S. Urbanek, M. Theus. JGR, A unified interface to R. *useR!*, 2004.

[20] J. Rainer, F. Sanchez-Cabo, G. Stocker, A. Sturn, and Z. Trajanoski. CARMAweb: comprehensive R- and bioconductor- based web service for microarray data analysis. *Nucleic Acids Research*, 43:W498-W503, 2006.

[21] C. Romualdi, N. Vitulo, M. D. Favero, and G. Lanfranchi. MIDAW: a web tool for statistical analysis of microarray data. *Nucleic Acids Research*, 33:W644-W649, 2005.

[22] M. Psarros, S. Heber, M. Sick, G. Thoppae, K. Harshman, and B. Sick. RACE: Remote Analysis Computation for gene Expression data. *Nucleic Acids Research*, 33:W638-W643, 2005.

[23] C. Smith. Textual Description of webbioc. http://www.bioconductor.org/repository/devel/vignette/webbioc.pdf.

[24] StatDataML http://www.omegahat.org/StatDataML/

[25] CDFML http://cdf.gsfc.nasa.gov/html/cdf_xml.html