Enabling R packages for web or grid services: lessons learned

Martin Morgan[*], Nianhua Li, Seth Falcon, Robert Gentleman,

16 February, 2007

1 Prelude: motivation

There are many reasons for exposing R packages as web or grid services. Main reasons motivating our work are as follows:

1. Provide standardized workflows. A well-defined web service simplifies and consolidates complex steps in an analysis into a single service. This standardizes the analysis so that it is reproducible in the hands of different users, including users with no R experience.

2. Aid interoperability. Web services require strongly typed service inputs and outputs, and (strive to) represent data in a language-neutral manner. Strongly typed data output from an R web service can be used as inputs to web services written in other languages.

3. Enhance access to powerful analytic methods. R methods and packages exposed as web services allow the unique strengths of R (e.g., statistical modeling) to be exposed to and accessed by other programming languages.

4. Access specialized computing resources. Web services separate the computing resources required of the client from those required by the server. This sets the stage for powerful computing resources to be accessed and shared by many users.

5. Centralize computing administration, while easing 'end-user' maintenance requirements. The often complex task of maintaining R, including regular updates to R itself and continual updates to availble packages, can be managed in a centralized fashion in a way that minimizes disruptions to the user's work.

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

6. Leverage Java resources. The large Java community has many active projects that help to effectively expose R as a web service. These Java resources range from the core functionality provided by tomcat to the messaging and queue management facilities of activeMQ.

7. Expose R statistical functionality to Java programmers. Easy facilities for producing web services from R packages via a Java intermediary means that the statistical computational abilities of R are more readily accessible to Java programmers.

2 Overcoming technical issues

A primary technical challenge to offering R packages as web services is to interface the R programming language with XML-based web services.

- We chose to target Java as the initial translation from R, rather than a more ambitious attempt to write web services functionality for R directly. Key issues include:
  - Pro: This approach provides access to mature Java web service resource tools.
  - Con: Typically, this introduces a data and service invocation translation layer (from R to Java, in addition to the translation to XML). This additional translation layer is not likely to impose a significant cost in terms of overall execution time, if only because parsing between native types and XML (a necessary step regardless of strategy) is typically very slow.
  - Con: The available object model (classes and methods) is reduced to the intersection of R, Java, and XML object models. This requires that certain R constructs (e.g., class unions) be employed with caution and that others (e.g., multiple inheritance) be avoided entirely.

- Web services and Java require strongly typed methods and well-defined data objects. The TypeInfo package provides facilities for strongly typing R functions and §4 methods. The §4 class system provides enough structure for well-defined data types. Both TypeInfo and the §4 system provide language introspection to programmatically translate R methods into strongly typed Java signatures.

  Even with these solutions, many R methods and classes cannot be easily represented in a way appropriate for web services. S3-style classes do not contain enough information for language introspection to determine mapping between R and Java types. The `list` type translates to Java `Object[]`, but this is not sufficiently rich for use in a web services context. A solution is to wrap such data objects as S4 classes.

  Conversely, a common data paradigm in Java or XML is a collection of objects of complex type `T`. While R might represent this as a `list`, with each member of the list implicitly of type `T`, TypeInfo does not provide an idiom for recognizing this paradigm and recovering appropriate information programmatically. A solution is to provide moderate type information in R (object of type `list`) and strong typing in Java. An alternative solution is to recast the data structure in a way that can be strongly typed, e.g., a list of numeric vectors might be represented as a numeric matrix.

- We use SJava and additonal facilities to accomplish R↔Java data and method mapping. RWebServices implements two different object models for base R types.

  The `robject` model more-or-less faithfully represents the underlying structure of R objects in Java (e.g., a 'matrix' is vector of data values, a vector of dimensions, a type label, facilities for 'names' and `NA` values, etc.).

  The `javalib` model is more faithful to Java data representations; a `matrix` must be typed as, e.g., *NumericMatrix*, and is represented in Java as, e.g., column-major `double[]` and associated dimensions `int[]`. There are no provisions for `NA` or R attributes such as `names`.

  These two different object models have consequences for interoperability (likely easier to achieve with the `javalib` model) and representation of statistical data (better with the `robject` model).

A second group of technical challenges revolve around service availability and evaluation.

- The architecture adopted separates Java-based service functionality from R / Java worker functionality. This means that R does not need to be available to the web server, simplifying deployment and risks of server-side exposure to nefarious activities.

- A realistic service model requires ability to manage multiple requests simultaneously. We use activeMQ to implement a messaging layer including customizable queues. activeMQ is deployed separately from the web service, e.g., inside a firewall. Computation is performed by R workers. Workers can be dynamically added to the pool, deployed on separate hosts, and customized to be capable of evaluating one or several services. Workers are persistent, minimizing service invocation costs.

Insights into additional technical challenges include:

- It is important to be able to conveniently encapsulate the service portion of RWebServices into other web service containers, e.g., using the introduce tool of caGrid. The architecture of the service side of RWebServices accomodates this separation.

- Statistical data offers unique challenges, for instance:

  - 'Missing' or `NA` values are distinct from non-computable (`NaN`) or not representable (e.g., `Inf`) values. These must be propagated successfully, both as input and return values. The `robject` model facilitates this (at the expense of greater client complexity, to continue the contract of dealing appropriately with `NA`); the `javalib` model assumes (at the risk of a runtime error) that any `NA` values are removed before service invocation (client responsibility) and before service return (R service responsibility).

- Web service methods require programmatic (e.g., brief method and class description) and user (e.g., detailed desription, interpretation of return values) documentation. RWebServices parses R `man` pages for method and class descriptions, annotating these as javadoc to provide programmatic documentation, Complete user-level documentation is only available inside the R package.

3 Adapting to a web services environment

The interative, exploratory aspects of R translate poorly to the stateless and high-latency web services environment. Lessons learned in addressing this issue include:

- Construct a coarse workflow granularity. Do this by identifying and consolidating common sequences of analytic steps, typically accomplished by arranging a sequence of R package function calls into a logcial workflow. Enhance the utility of the workflow by selectively exposing parameters available for manipulation – this represents the transition from R research software to web-based service.

- Simplify result types. Many R functions rely on side-effects (e.g., plots), but these are not useful for subsequent computation. Detailed results are sometimes only useful within R. In these cases it is appropriate to simplfy result types to emphasize computable data.

- Imprimatur of scientific authority. R's pre-eminence as a research tool means that exploratory or experimental methods may be implemented, but these are often not appropriate for general or uncritical use. The services exposed need to be vetted to include only scientifically sound and established methods.

4 Future opportunities

Lessons learned during this project point to several future opporutnities.

- Implementing stateful services represents an opportunity to reduce data latency and restore some sense of interactive analysis. For instance, stateful services might facilitate services returning data for subsequent analysis, and services for return of non-computable results like plots.

- Separating analytic services from the clients using them places an interpretive burden on the client. For instance, an R user might combine input and output data into a figure, and use this to visually assess and guide subsequent analyses. This requires knowledge about how to appropriately superpose input and output data, in addition to the tools to do this. These tools are implicit in R, but must be made explicitly available in the client. Possible solutions include:
  - Burden lies with client. This solution requires that the client be programmed to interpret results, rather than merely retrieve them.
  - Service returns complex data objects (e.g., a graphical summary of input and output, in addition to output data). The client can access parts of the object as appropriate for subsequent workflow, but needs to decompose the returned structure appropriately. Sufficiently complex return types could be difficult to document in a semantically meaning way.
  - User interacts repeatedly with stateful services. This solution requires that the client maintain a sense of state, and offers the user an indication of dependencies amongst services (e.g., viewing a plot only makes sense after an analysis has been performed).
- Documentation. Existing documentation tools and requirements emphasize programmatic descriptions of the API (e.g., javadoc) or (in a caGrid context) semantic classification of arument and return types. This level of documentation is inadequate for the user, who requires access to full manual pages for methods or tutorial-like documents summarizing appropriate use of functions.