Author's Responses to Review of

JSS 1187

Original Title — **jvmr**: Integration of R, Java, and Scala

New Title — **rscala**: Integration of R and Scala

David B. Dahl
Brigham Young University

May 15, 2015

Associate Editor: Comments and Responses

Comment 1: *I am sympathetic to the goals of the project described in the software. Unfortunately, neither the paper or the software by themselves is very compelling. If either was compelling, we could potentially accept the submission. So either the software should be improved (in its design and computational model - see below), or the paper should be refocused to include a general and high-level argument about why and how the languages covered by the package should be used together to improve the state of statistical/scientific computing. Ideally both the paper and the software should be improved. Neither is particularly hard for experts/advanced researchers in statistical computing. The general ideas and designs have been around for other interfaces for 15 years! Furthermore, the case studies should be more compelling, authentic and illustrate new things we can do because of the use of both languages, rather than showing how to do something we can easily do already in any one of the languages. As an alternative to resubmitting an updated version of this paper, the authors might consider submitting a version to the R Journal or another journal.*

Response:
Thank you for the opportunity to resubmit. Based on your feedback and the feedback of the other reviewers, we have addressed issues with both the software and with the paper. In summary, we have: i. rewritten the software to greatly improve the design, functionality, and computational model and ii. refocused the paper to better motivate the software, define the audience, streamline the presentation, and provide more compelling and authentic examples

and case studies. In redesigning the software and refocusing the paper, we decided to remove support for BeanShell as it as a distraction from the main contribution of the paper: Integration of R and Scala. Accordingly, we also renamed the package. Details on all of the changes are summarized in the responses to the referees.

We appreciate the suggestions for improvement and feel that the revised paper is stronger as a result. Thank you.

Reviewer #1: Comments and Responses

Summary of Response:

Based on your feedback and the feedback of the other reviewers, we have addressed issues both with the software and with the paper. In summary, we have: i. rewritten the software to greatly improve the design, functionality, and computational model and ii. refocused the paper to better motivate the software, define the audience, streamline the presentation, and provide more compelling and authentic examples and case studies. In redesigning the software and refocusing the paper, we decided to remove support for BeanShell as it was a distraction from the main contribution of the paper: Integration of R and Scala. Accordingly, we also renamed the package. Details on all of the responses to your comments are below.

We appreciate the suggestions for improvement and feel that the revised paper is stronger as a result. Thank you.

Comment 1: *This paper is fairly clearly written and describes an approach to programming for data analysis that can employ multiple languages. Table 1 depicts some syntax related to 'primitives', vectors and matrices, but does not address the definition and evaluation of functions in the different languages. This might be worthwhile. To make room, one might drop the Vectors column of the present table.*

Response:

Based on this feedback and others, we redesigned the software to accommodate the definition and evaluation of functions. See Section 2.2 and Table 1. (The Table 1 that you mention now corresponds to the new Table 2.) New subsections that are especially pertinent to your suggestion are titled: "Instantiating Scala objects and calling methods", "Return values when calling Scala", "Arguments when calling Scala", and "Defining inline Scala functions".

Comment 2: *I personally do not understand the use of the* `${1}` *notation in the scala calls in table 2, so it seems to make sense to address this early on. [We see that there are up to 9 string substitutions permitted, mid page 9 – 'numerical arguments are converted to strings...' This aspect of the interface should be discussed in detail with respect to pros and cons. Is this a limitation to be relaxed in later development?]*

Response:

Your comment caused us to rethink this feature. We removed the limitation so that any number of string substitutions are permitted. Moreover, we redesigned the interface to make it more useful. We have replaced the `${1}` notation with much more general string interpolation using the `@{...}` notation, where `...` is any R expression. This is described at the end of the subsection "Evaluating arbitrary Scala code" in Section 2.2.

Comment 3: *Section 4 is lengthy and not very demonstrative of high value. String variables can be concocted in scala to define R programs which are then evaluated in the embedded R. Variable values can be pushed and pulled between different systems. This section can be shortened considerably with details and examples relegated to an appendix. The operational details are not important, but the concepts and efficiencies for software creation and reuse are at a premium for those who will study this paper, and this is where we need to put the energy.*

Response:

We agree that the original Section 4 was too long relative to its value. We have completely removed this material. Its closest analog in the current paper is found in parts of Sections 2.2 and 2.3. We believe that the paper now better motivates the software and conveys the important concepts and efficiencies. We emphasize the ability to seamlessly integrate Scala and R in a single project and the code that remains in the body of the paper helps convey concepts. Readers can gain more details by studying the new examples and case studies whose code is available in the online supplement.

Comment 4: *Section 5.1 is not very satisfying. A web site framework has an array of arrays of doubles named births, and the R.update and as.data.frame are assumed to convert this into a structure suitable for ANOVA. The names of the variables are assigned by the R client function. Is there no introspection available for the Play-based data to tell the variable names and allow some validation of the data before the task is attempted? This example shows nothing over the methods elaborated in section 4 (which I do not think are very demonstrative anyway) and should be enriched to show how good interoperation between scala and R lead to an effective approach for adding statistical value to web/data interfaces.*

Response:

We agree and have removed that example. Among the several new examples and case studies is a more compelling web application developed with the Play Framework. The new example uses rscala to leverage existing R packages to display historical temperature data. The example shows data validation and interoperation between Scala and R. This is described in Section 5.2 and the complete case study with instructions and code is available in the online supplement.

Comment 5: *Section 5.2 begins with a typo (birth for births) and is now using f(x[x[,2]==2,1],...) to select the information of interest. It probably works in this specific situation but why would a "developer" want information on the "second herd"? This is about concepts underlying tools and the fact that we can make such references and conversions is pretty obvious.*

Response:

We agree and have removed this example.

Comment 6: *Section 5.3 is potentially much more interesting. The scala interpreter is applied to a long string representing Bayesian logistic regression. A 15x speedup relative to a certain* *version of R is noted. The data structure is very small, so one cannot infer that much from this example. One would tend to use multiple chains in parallel for a problem like this, and the code for making this extension in R is quite simple – taking this example much more seriously seems to be a potentially useful direction, paying attention to generality of interface, adaptability to various environments (multicore vs cluster), and all aspects of fairness or unfairness of the comparison.*

Response:
Thank you for this feedback. Our new Section 5.1 incorporates your suggestions and presents a more extensive investigation using an MCMC algorithm for Bayesian logistic regression. We now consider two datasets, one somewhat larger than our original dataset and another significantly larger. We also investigate the performance using both single chains and multiple chains in parallel. The statistical model itself is more realistic and the posterior distribution is now bivariate instead of univariate. We benchmark several implementations of the algorithm. The complete code for this case study is available in the online supplement.

Comment 7: *In summary, this paper describes interfaces between languages that may have impact on practicing data analysts, particularly in the domain of software reuse and strategy selection for tool design. But in its present form it does not seem to make anything like the strongest arguments possible for the choices made in rjvm, and would not in my view be a compelling argument for a developer moving to master scala as an important element of the statistical computing toolkit. The efforts made by the authors suggest that they believe that it and the more general topic of JVM-based software SHOULD be regarded as important elements of statistical computing, so I would request that the paper be revised to exhibit good arguments in this vein, putting aside details of syntax.*

Response:
Thank you for your suggestions for improving the paper and software. The new paper now makes it clear who the intended audience is: users who have already made an investment in R and Scala or who are at least considering it. We now have a much better software design and the paper provides a more compelling presentation and examples. We do, however, stop short of explicitly advocating Scala for statistical computing in the paper. There are a variety of languages that one might consider to complement R and we feel that it would be beyond the scope of this paper to try to convince someone to change from their favorite language. Nevertheless, there is a substantial community using Scala and a substantial community using R. We hope that rscala lowers the barriers between them. In summary, we have: i. rewritten the software to greatly improve the design, functionality, and computational model and ii. refocused the paper to better motivate the software, define the audience, streamline the presentation, and provide more compelling and authentic examples and case studies.

Reviewer #2: Comments and Responses

Comment 1: *The paper discusses "connecting" R and 3 other languages that run on the Java Virtual Machine - namely, Scala, Java and the BeanShell (an interactive Java-like language). The software may be useful and this is what makes an enhanced version of the paper and software suitable for publication. However, the computational model and design of the software are not particularly compelling (see below). Furthermore, the paper does not present compelling reasons for why readers would find the interfaces valuable. The paper is not unreasonably structured, but fails to make a convincing case. It focuses on many minute details about how to install the software and get it running, and covers the basic syntax but not the computational model and how to think about using the connections between the languages, i.e. the larger picture. There are 4 short case studies - 3 of these are okay, but far from compelling. Example 5.3 is somewhat interesting, but still leaves lots of questions as to why one would want to use this particular pair of languages. Most importantly, there is no discussion of the pros and cons of the approaches and comparison with other approaches. As such, the paper focuses on simple aspects and ignores the more complex issues.*

Response:

Based on your feedback and the feedback of the other reviewers, we have addressed issues both with the software and with the paper. In summary, we have: i. rewritten the software to greatly improve the design, functionality, and computational model and ii. refocused the paper to better motivate the software, define the audience, streamline the presentation, and provide more compelling and authentic examples and case studies. These examples and case studies provide a vehicle to discuss the advantages and disadvantages of our software, both in terms of code reuse, convenience, and computational efficiency. In redesigning the software and refocusing the paper, we decided to remove support for BeanShell as it as a distraction from the main contribution of the paper: Integration of R and Scala. Accordingly, we also renamed the package. Details on all of the responses to your comments are below.

We appreciate the suggestions for improvement and feel that the revised paper is stronger as a result. Thank you.

Comment 2: *The computational model for the inter-system/language interfaces is quite simple-minded and much less rich than many existing and common language interfaces. Essentially, it requires the user to program in both languages, e.g. Scala and R. It requires, for example, Scala users to specify R commands as strings to the R evaluator. This is awkward, at best. It requires users to serialize local data in Scala into the R command, or to explicitly push local data across the interface to global variables in the R session and then refer to these global variables in the R command. Both of these require unnecessary work by the user: thinking in two languages and managing two work/namespaces. It also uses global variables which is simply undesirable from a software engineering perspective.*

Instead, of the current approach in jvmr, a much richer model is to allow allow one to call, e.g., R functions from within a Scala computation by passing objects in the local language (Scala) directly as arguments to that function, e.g. R.foo(arg1, arg2, arg3) This is relatively *straightforward to implement, but is a very different interface. Indeed, it is a superset of the jvmr interface and has been advocated from as early as 1999 in the original Java interface - SJava.*

Response:

This is an accurate description of the original software. Thank you for the constructive criticism on the use of global variables. The package now uses its own private environment in which R code is evaluated and values are assigned and retrieved. Moreover, we have taken the feedback seriously and have completely redesigned and rewritten the software to improve the computational model.

We are particularly pleased with the interface embedding Scala in R. Taking inspiration from the rJava package, our rscala package now transparently brings Scala methods to R as if they were native R functions. Specifically, Scala classes can be instantiated and methods of Scala classes can be called directly. In addition, inline Scala functions can be defined and callbacks to the original R interpreter are supported. We retain the ability to execute arbitrary Scala code on the fly from within R.

Since Scala and Java are precompiled, statically-typed languages, R's dynamically typed functions cannot be brought into Scala and Java as if they were native methods. Although rscala does not provide an API to access the internals of an R object (i.e. SEXP), the package does provide a scripting interface allowing the evaluation of arbitrary R code. This scripting interface is similar to that of Rserve and rJava's JRI. Convenience methods for getting and setting values are provided that avoid explicit casting. It would be nice to permit code in Scala like `R.foo(arg1,arg2,arg3)`, but we believe that this is more suitable for dynamically-typed languages and not for precompiled, statically-typed languages like Scala and Java. The Scala compiler cannot know at compile time the return type of the dynamically-typed `foo` function in R. Using a general type, such as Scala's `AnyRef` (which is equivalent to Java's `Object`), as the return type is often not helpful because the result needs to be cast before it can be used in most contexts. Hence, we prefer code such as `R.evalD1("rnorm(10)")` which evaluates the R snippet and tells the compiler that the result is an array of doubles.

Comment 3: *Furthermore, the software does not use the available reflection to determine the corresponding data types and so requires the user to explicitly identify the target type when transferring data from one side of the interface to the other. While the user should be able to specify control this, the obvious defaults should simply work. If the reflection is not available in Scala due to compilation, this can be discussed.*

Response:

Consider the two directions separately. When Scala is embedded in R, there is no need to explicitly identify the target type. This is illustrated with the following R transcript:

```
> s <- scalaInterpreter()
> storage.mode( s %~% 'Array (1,   2,   3  )' )
[1] "integer"
> storage.mode( s %~% 'Array (1.0, 2.0, 3.0)' )
[1] "double"
```

The storage modes for these results are 'integer' and 'double', respectively, but the user does not have to explicitly indicate this in the R code.

The situation is quite different in the other direction since Scala and Java are precompiled, statically-typed languages. When transferring data from embedded R in a Scala or Java program, the `get` method can be used. Although the run-time type is as it should be, the compile-time type is generic and the result will have to be cast before it can be used in most contexts. For convenience, we provide a series of methods (e.g., `getD0`, `getS1`, etc.) to specify the return type without explicit casting. Full details are available in Scaladoc and Javadoc found in the online supplement.

Comment 4: *Importantly, the paper needs to provide convincing arguments as to why integrating any pair of these languages is likely to yield significant improvement over the current state of affairs. The MCMC example is a positive step in this direction, but it does not address many issues that show it is an improvement over several other possible approaches. The other examples would be more compelling if they showed new things that are possible by using the two languages together rather than showing how to do something we can already do in any one of them alone.*

Response:

This point is well taken. We now clearly state the motivation for our package. We have significantly improved the case studies and now provide entirely new examples to make a more compelling case. In short, our rscala package aims to do for Scala what rJava has done for Java. Quoting from the new introduction, "We believe that Scala deserves consideration when looking for a general-purpose programming language to complement R, but advocating Scala for statistical computing is beyond the scope of this paper. Instead, this paper introduces our rscala package to those that are already somewhat familiar with R and Scala. The package allows users to seamlessly incorporate R and Scala (or Java) code in one program and utilize each language's respective strengths (including libraries, methods, speed, graphics, etc.) ... Thus, the rscala package allows an R developer to reuse Scala code and apply their Scala skills to make R extensions. On the other hand, rscala allows a Scala developer to make use of R's broad array of data analysis and graphing capabilities from within a Scala application. The rscala package is intended to provide this bridge to those who have an interest in both Scala and R. This is not unlike what has already been done with R and other languages ..."

Comment 5: *The paper does not address debugging across the interface, or how errors are propagated appropriately.*

Response:

In response to this comment, we have added Section 2.4, "Debugging and error propagation."

Comment 6: *The following are some notes I took as I read the paper. Who is the audience - for the paper? for the software itself? What level are you assuming?*

Response:

We now state that the audience is "those that are already somewhat familiar with R and Scala . . . The package allows users to seamlessly incorporate R and Scala (or Java) code in one program and utilize each language's respective strengths (including libraries, methods, speed, graphics, etc.) . . . The rscala package is intended to provide this bridge to those who have an interest in both Scala and R."

Comment 7: *Much of the paper (and importantly the first 2/3) reads like a lot of minute details (and they break the flow), but no overarching perspective or theme.*

Response:

The paper has been rewritten to address this concern. The new paper flows much better and has a sharper focus. In rewriting the paper, we took care to better motivate the software, clearly define the audience, streamline the presentation, and provide more compelling and authentic examples and case studies.

Comment 8: *The programming model/user interface feels like it was built to work, but is not very general or well designed. There are more general and better interface models that have been in existence for a long time. Furthermore, much of what is provided in the package can be built directly on the rJava package. R can be embedded directly within a JVM instance and hence Scala, BeanShell, and Java. The value added here is specializing this to Scala and the Beanshell. This is good, but the case needs to be made why this is helpful with realistic and compelling examples.*

Response:

We have improved the original programming model/user interface. We have completely redesigned and rewritten the software to be more functional and easier to use. We are particularly pleased with the interface embedding Scala in R. Taking inspiration from the rJava package, our rscala package now transparently brings Scala methods to R as if they were native R functions. Specifically, Scala classes can be instantiated and methods of Scala classes can be called directly. In addition, inline Scala functions can be defined and callbacks to the original R interpreter are supported. We retain the ability to execute arbitrary Scala code on the fly from within R.

Since Scala and Java are precompiled, statically-typed languages, R's dynamically typed functions cannot be brought into Scala and Java as if they were native methods. Although rscala does not provide an API to access the internals of an R object (i.e. SEXP), the package does provide a scripting interface allowing the evaluation of arbitrary R code. This scripting interface is similar to that of Rserve and rJava's JRI. Convenience methods for getting and setting values are provided that avoid explicit casting. It would be nice to permit code in Scala like `R.foo(arg1,arg2,arg3)`, but we believe that this is more suitable for dynamically-typed languages and not for precompiled, statically-typed languages like Scala and Java. The Scala compiler cannot know at compile time the return type of the dynamically-typed `foo` function in R. Using a general type, such as Scala's `AnyRef` (which is equivalent to Java's `Object`), as the return type is often not helpful because the result needs to be cast before it can be used in most contexts. Hence, we prefer code such as `R.evalD1("rnorm(10)")` which evaluates the R snippet and tells the compiler that the result is an array of doubles.

We discuss rJava in several places, including Section 4.1, "Comparison to rJava". There we note, "Since Scala compiles to Java bytecode and runs on the Java Virtual Machine, some of the functionality in the rscala package can be replicated using the rJava package. Both rscala and rJava can call precompiled Java bytecode. Features that rJava does not support include defining inline functions (Section 2.2.4), evaluating arbitrary code (Section 2.2.5), and explicit support for Scala. We now consider the difficulty of calling Scala from rJava. After working on several projects using rJava to access Scala code in R, we found that Scala was cumbersome to use. Scala provides several features that do not map directly to Java equivalents. The Scala compiler uses name mangling, behind-the-scenes code generation, and other techniques when compiling to Java bytecode. Calling Scala code that makes use of these features from rJava requires some understanding of Scala's compiler at a minimum, but often the developer is required to write a Java-friendly wrapper method in Scala that hides advanced Scala features."

In redesigning the software and refocusing the paper, we decided to remove support for BeanShell as it was a distraction from the main contribution of the paper: Integration of R and Scala. Accordingly, we also renamed the package.

Comment 9: *Unfortunately, the examples are quite simple and uncompelling and the discussion around these does not debate the advantanges and disadvantages of these and other possible approaches. There is very little motivation in the early parts of the paper to suggest why one would want to use this.*

Response:

In response, we removed the old examples and replaced them with more compelling examples and case studies. These examples and case studies provide a vehicle to discuss the advantages and disadvantages of our software, both in terms of code reuse, convenience, and computational efficiency. In particular, see Sections 4 and 5.

Comment 10: *The wording is imprecise and this reduces my confidence in the work. example is types used for boolean, integer, etc. and also for vector, matrix, which are data structures. p3 "The jvmr package supports four data types (booleans, integers, doubles, and strings) in three types (primitives, vectors, and matrices)."*

Response:

Thank you. We are more careful in describing types in the revision. We note here that the term "type" has a broad meaning in Scala. For example, both `Int` and `Array[Int]` are referred to as types.

Comment 11: *The introduction (first 4 pages) is fact related and details but doesn't convey the ideas. Setting environment variables, and minute-level how-tos...*

Response:

This is a fair point. We believe that our rewritten paper addresses your concern, including a better-motivated introduction and streamlined Section 2, “Usage of the package”.

Comment 12: *I think there should be more of a general introduction and some brief motivating examples. The intro. just states you can use any of the languages. But it doesn't illustrate why you would want to do that, except in very vague terms (e.g. functional and OOP paradigms). Since R has both, this seems a little incomplete, to say the least.*

Response:

Thank you for the suggestion. We believe that our new introduction is much improved. In short, “The rscala package allows an R developer to reuse Scala code and apply their Scala skills to make R extensions. On the other hand, rscala allows a Scala developer to make use of Rs broad array of data analysis and graphing capabilities from within a Scala application ... The rscala package aims to do for Scala what rJava has done for Java.”

Comment 13: *P4 reads like a users manual, or more specifically, a ”Getting Started”/installation guide and is not particular good for that either. This doesn't seem relevant to an academic paper. It is the wrong level, in my opinion.*

Why is it not good even for a users manual. One example - ”Windows users should use a semi-colon... + next sentences”. This is so detailed and not clear. Just show the Windows version.

There is no explanation of the 3 lines of code that compile and invoke scala.

These 3 lines are a mix of shell and make commands apparently There seems to be confusion in the authors' minds.

Still no example but just vague and general talk of a scala REPL.

Response:

These are fair points and we appreciate the feedback. We have rewritten the paper to incorporate these and other suggestions. Our new Section 2 quickly introduces our software to users of Scala and R so that subsequent sections can build on the ideas, discuss the underlying implementation, compare against other software, and demonstrate case studies.

Comment 14: *P4 You don't explain how the multiple interpreters are implemented - e.g. separate processes and sockets, or embedded? That's okay not to do this here, but it is of interest and it would be nice to have had a brief technical introduction to the design/implementation earlier.*

Response:

Thank you for your feedback. We have incorporated your suggestions into our new Section 3, “Implementation”.

Comment 15: *Sending R commands from Scala to R is a very weak model. Who wants to program in two languages. Intellectually, it is the most obvious and least powerful. It is fine for simple interaction, but not for building complex software. Debugging is a serious issue. Using two name/workspaces puts a burden on the...*

Response:

We have improved the original programming model/user interface. We have completely redesigned and rewritten the software to be more functional and easier to use. We are particularly pleased with the interface embedding Scala in R. Taking inspiration from the rJava package, our rscala package now transparently brings Scala methods to R as if they were native R functions. Specifically, Scala classes can be instantiated and methods of Scala classes can be called directly. In addition, inline Scala functions can be defined and callbacks to the original R interpreter are supported. We retain the ability to execute arbitrary Scala code on the fly from within R.

In response, we have also added Section 2.4, "Debugging and error propagation."

Comment 16: *Sending R commands from Scala to R is a very weak model. Who wants to program in two languages. Writing R code in Scala means you don't have syntax highlighting, tab completion. So how does one deal with these issues?*

Response:

Since Scala and Java are precompiled, statically-typed languages, R's dynamically typed functions cannot be brought into Scala and Java as if they were native methods. Although rscala does not provide an API to access the internals of an R object (i.e. SEXP), the package does provide a scripting interface allowing the evaluation of arbitrary R code. This scripting interface is similar to that of Rserve and rJava's JRI. Convenience methods for getting and setting values are provided that avoid explicit casting. It would be nice to permit code in Scala like `R.foo(arg1,arg2,arg3)`, but we believe that this is more suitable for dynamically-typed languages and not for precompiled, statically-typed languages like Scala and Java. The Scala compiler cannot know at compile time the return type of the dynamically-typed `foo` function in R. Using a general type, such as Scala's `AnyRef` (which is equivalent to Java's `Object`), as the return type is often not helpful because the result needs to be cast before it can be used in most contexts. Hence, we prefer code such as `R.evalD1("rnorm(10)")` which evaluates the R snippet and tells the compiler that the result is an array of doubles.

It is true that syntax highlighting and tab completion for R code embedded within Scala is not currently available in Scala development environments. In practice we have found it convenient to develop R code snippets in R's REPL and/or a scratch file and then copy it into Scala source code.

Comment 17: *Using `R> "words"` to access the R variable words seems a little bizarre. In R, we use words, i.e. the variable name or _symbol_. Here we are using the literal string. We are using R syntax, but not using R syntax. The computational model is not necessarily intuitive* *and conssitent.*

Response:

This is a fair point. We have removed the “>” operator and now provide a more intuitive and consistent computational model.

Comment 18: *How is apply similar to eval ?? Why does eval not return an object? What's the benefit of this?*

Response:

The comment on `apply` versus `eval` relates to our old computational model which has been removed from the software. On the second point, the `eval` method of the `RClient` class takes two arguments: the R expression to evaluate and a boolean indicating whether the result should be serialized over the TCP/IP sockets. If the result is not needed, avoiding this serialization saves time.

Comment 19: *Talk of Scala type Any but with no background to Scala - who is the audience?*

Response:

We have removed the unnecessary reference. In response to your comment and those of the other referees, we now are clear that the audience is “those that are already somewhat familiar with R and Scala . . . The package allows users to seamlessly incorporate R and Scala (or Java) code in one program and utilize each language's respective strengths (including libraries, methods, speed, graphics, etc.) . . . The rscala package is intended to provide this bridge to those who have an interest in both Scala and R.”

Comment 20: *Why does the user have to specify the type of the remote object when it can be determined dynamically. The example at the bottom of page 5 shows the redundancy* ***`println(R.toPrimitive[Int]("as.integer(n)"))`***

Response:

Consider the two directions separately. When Scala is embedded in R, there is no need to explicitly identify the target type. This is illustrated with the following R transcript:

```
> s <- scalaInterpreter()
> storage.mode( s %~% 'Array (1,   2,   3  )' )
[1] "integer"
> storage.mode( s %~% 'Array (1.0, 2.0, 3.0)' )
[1] "double"
```

The storage modes for results are ‘integer’ and ‘double’, respectively, but the user does not have to explicitly indicate this in the R code.

The situation is quite different in the other direction since Scala and Java are precompiled, statically-typed languages. When transferring data from embedded R in a Scala or Java program, the `get` method can be used. Although the run-time type is as it should be, the compile-time type is generic and the result will have to be cast before it can be used in most contexts. For convenience, we provide a series of methods (e.g., `getD0`, `getS1`, etc.) to specify the return type without explicit casting. Full details are available in Scaladoc and Javadoc found in the online supplement.

Comment 21: *There is no description of the update() method but just an example that describes it by equivalency to assignment. This is an example of where the ideas are not clear and not well explained. Why do we need both approaches? How are they different? If they are not, why have both? Is it just syntactic sugar.*

Response:

We redesigned the software based on feedback and this issue no longer exists. We now provide a consistent interface that is explained in the paper and documented in Scaladoc and Javadoc found in the online supplement.

Comment 22: *Furthermore, the [operator is a strange one for evaluating an expression. In what sense is an intepreter subsettable. I understand that it is related to accessing variables and so a["expression"] and a["variableName"] are similar. However, $ or [[is more appropriate for accessing _individual_ variables in the remote system.*

Response:

This is a good point. In response, the package now uses $ to access and assign individual variables. Please see the subsection “Getting and setting values” in Section 2.2.

Comment 23: *What does "open an R prompt" mean? And why would a user want it? Also, one can do this reasonably easily in an IPython notebook with code for two or more languages. Then all the caveats about the warning that might appear. This indicates a) the paper is way too specific, and b) a lack of clean design that resolves this problem.*

Response:

We agree that other solutions may be better suited and we have removed the “R prompt” functionality.

Comment 24: *Example 5.1: While I like the idea of using a non-R framework to handle the Web server passing information to the script, the example glosses over this and does not explain the overhead of learning this different setup. Furthermore, it doesn't compare this to RApache, CGIwithR or other R-based approaches. As such, the example is not compelling. Since the entire purpose of the scala script is to marshal the data to R to do the real computations, it is much more direct to use an R-based approach to avoid the extra unnecessary Scala layer. Again, there is no discussion of the pros and cons.*

Response:

There are certainly many ways in which R could be incorporated into a web application and we mention several in the paper. We provide details on one such example in our all-new Section 5.2, "Scala web application using R". There we suppose ". . . a Scala developer maintains a website based on the Play Framework for web development. He is tasked with adding new functionality to the website . . . He can look for a Scala or Java library to [accomplish the task]. If he is already familiar with R, however, the data, computation, and plotting tasks are easily accomplished [with existing R packages] . . . The task for the developer now becomes how to serve the plot and information in the Play Framework. Two possible solutions are to use Rscript or Rserve, as discussed in Section 4.4.3. Moving away from the Play Framework, one might consider setting up a web server using RApache (Horner 2013), CGIwithR (Firth 2003), or Shiny (Chang and et al 2015). In this case study, however, we assume that one has already invested resources in setting up and maintaining a website based on the Play Framework. We demonstrate how easy it is to add the desired functionality to the existing infrastructure using rscala."

Comment 25: *Example 5.2: This is not very interesting. Why would one want to do this? Use an authentic example.*

Response:

We agree. We have removed this example and have replaced it with more authentic examples and case studies.

Comment 26: *Example 5.3: This is the best example. It could and should be developed more. Also, it is important to compare it to existing facilities in R, e.g. JAGS, BUGS, Stan, mcmcpack, ... Once the code in the remote system becomes in any way complex, i.e. more than a line or two as in the examples in most of the paper, issues of debugging and error handling are important. This is not addressed in the paper.*

Response:

Thank you for this feedback. Our new Section 5.1 incorporates your suggestions and presents a more extensive investigation using an MCMC algorithm for Bayesian logistic regression. We now consider two datasets, one somewhat larger than our original dataset and another significantly larger. We also investigate the performance using both single chains and multiple chains in parallel. The statistical model itself is more realistic and the posterior distribution is now bivariate instead of univariate. We benchmark several implementations of the algorithm. The complete code for this case study is available in the online supplement.

With regard to other facilities, we write "In this section we use a custom Markov chain Monte Carlo (MCMC) algorithm to fit a Bayesian logistic regression model in R. This model can also be estimated in R through a variety of other algorithms using R packages such as . . . Our interest here, however, is not to determine the best algorithm for Bayesian model fitting, but rather to compare the convenience and computational speed of Scala, C, and standard R in implementing the same commonly-used, computationally-expensive algorithm."

We also added Section 2.4, “Debugging and error propagation.”

Comment 27: *Example 5.4 is almost an illustration of the wrong way to think about this bridge. The goal is to exploit a Java library in R. However, there are almost as many lines of Java code than R code in the example. The only reason there are fewer is because...*

Response:
We agree and have removed this example. We believe that the new examples and case studies are more authentic and compelling.

Comment 28: *The functionality may be useful. But there is no explanation/motivation of why/how. Why would you want to use the R prompt functionality and then switch back to scala?*

Response:
Upon reflection, we found the “R prompt” functionality to be of little value and have removed the feature.

Reviewer #3: Comments and Responses

Comment 1: *It seems that jvmr uses the registry on Windows to locate R and therefore can't find R on my system since I keep my registry clean. Can you ask them to fix this by, for example, having an environment variable that one can set to specify the location of R (e.g. `R_HOME=C:\Program Files\R\R-3.0.2`). It can still look in the registry if that environment variable is not set. Thanks.*

Response:
It is true that the package uses the registry to locate R on Windows by default. The option "Save version number in registry" is enabled in a default R installation on Windows but, as you point out, some users may wish to disable this option. Thank you for your suggestion. We have modified the `apply` function of the `org.ddahl.rscala.callback.RClient` companion object to provide an argument `rCmd` which can be overridden to specify the path to R, e.g.:

```
val R = org.ddahl.rscala.callback.RClient("C:/Program Files/R/R-3.1.3/bin/R.exe")
```

We now explain this in the paper.