A Brief Overview of R and Object-Oriented Programming

Shawn Mace
Middle Tennessee State University

1. INTRODUCTION

It was by chance that Robert Gentleman and Rob Ihaka met in 1992. Gentleman was a professor University of Waterloo in Canada who travelled nearly 9000 miles for a three-month speaking engagement at University of Auckland in New Zealand. While there, Gentleman, a statistician by trade, was working on a statistical problem and needed assistance with a complicated program he needed to execute and check some particularly involved computations. The only person to whom Gentleman could turn at the university was Ihaka, who happened to have a manual for the program. The two soon learned that they shared similar interests, namely both had an interest in statistical computing and recognized there was a need for a better software environment to handle advanced statistical analysis - Gentleman for his own works and Ihaka, then a professor of statistics, for his tools to use in his teaching laboratory. [1] Neither knew a suitable suite or program to fit their needs.

Both Ihaka and Gentleman shared a knowledge of Scheme, and both found the language useful in a variety of ways. Scheme, however, had difficult and challenging syntax and lacked desired functionality. S, the main statistical program of the era, provided the kind of syntax they were looking for but lacked portability – Ihaka's laboratory operated on Macintosh that could not use it. This idea was not a new epiphany for Ihaka. He once, before meeting Gentleman, attempted to demonstrate an idea of his stating that lexical scoping could be used to obtain own variables – in Rs case, these tests where the forerunner of the basic R objects obtained through variables like arrays, matrices, vectors, lists and dataframes. He did not have a copy of Scheme available, so he attempted the demonstration for a colleague in S. The attempt failed due to scoping differences, but it set Ihaka's mentality squarely in how S could be augmented to fulfil his desires. There was no such program that met his desired harmonious synergy of Scheme and S, so the gears began to turn.

The first attempt to see the project to fruition was met with complications. Ihaka reflected on his earlier attempts reflecting that his first attempts at building a compiler was gleaned from LISP and consisted of roughly 1000 lines of C code, where data structures supporting the work he desired along with a new attempt at a user-interface. In August of 1993 following many calls for S that worked on Macintosh systems and the revealed an apparent fertility for their idea in the industry of statistical programming, Gentleman and Ihaka made plans to release R, which was named as an homage to S and the first letter of both of the creator's name. [2] Soon after, a usable binary version of R appeared on StatLib, an online system for distributing statistical software and data, which drew R out of the small crevices of Auckland anonymity to a much wider audience.

2. PHILOSOPHY

In 1996 Gentleman and Ihaka explored commercial avenues for R. They decided the best use of their work would be occur if they made R available for free. Upon admittance to the GNU library they released R to the open-source market at the same time creating R-Core, a group of 20 programmers who perform official up-keep and updates on the system. This spawned CRAN, Comprehensive R A Network, which houses, updates, and maintains manuals, guides, and download links of official and community-built packages and suites that meet their stringent criteria and standards.

Ihaka and Gentleman tried to keep R small using mark-and-sweep garbage collecting and reference counting to keep memory demands low. This had a side-effect of making the interpreter fast, notably during loops and in array mutations. R is ultimately designed to be portable working with any ANSI C compiler. The syntax is familiar, yet under the hood it more resembles Scheme by abandoning the idea of call-frames in favor of lexical scoping to provide a better way to preserve function state when going from function to function.[4] At the heart of R lay a yearning to assist in the promotion of platforms to knowledge, an ideology held by the vibrant R community to freely assist their fellow R users.

Building upon those cornerstones, R has become one of the most popular statistical programs in computing. What separates R from SAS or SPSS (two other popular and effective languages in the field; SAS is the largest privately ran software enterprise and SPSS is implemented through IBM) is that R is open-source whereas the former are for-profit and can prove unavailable for many that could utilize it in an academic circles or those that were curious. This has allowed for many adaptations which improve the efficiency and flow of R, as well as offer a large database of user-defined packages and very active user support groups. R has been used by Google to handle any data manipulation and visualization, implemented by BioConductor to analyze genomic data. Other companies that have adopted the technologies range from Google, Twitter, and The New York Times. [3] R has found a home in financial, sporting, and manufacturing designs as well.

3. INFLUENCE

3.1 Scheme and S

R is heavily influenced by S and Scheme. Gentleman and Ihaka wrote a scheme-like interpreter, easy enough for them given the readily available resources that have directly mapped the process. [4] To supply all the changes they wished to visualize they wrote the program and on-board compiler (though through their GNU

connections GNU's compiler, since it was partially written in C, could be used) to provide the requirements which creating a vast amount of functional capabilities still in use in modern versions of R.

R is a sublanguage of S. S was created by John Chamber. Chambers who authored the seminal book on S language entitled "Statistical Models in S". He was also head of advanced software development at Bell Laboratories. He once wrote that he developed S because he wanted to develop an interactive environment that users didn't have to think of as programming. [7] R used the syntax of S. The semantics resemble S but are more functionally and practically akin to Scheme.

R was designed to run on every system whether Mac, Windows, Linux, or even gaming consoles like Xbox or PlayStation. This diversity lends itself to active development, both by R-Core (of whom Chambers is now a member) and the R-community. Before the release of R 2.0, the community had already shown its merits. Some of the early packages (most created by Vienna graduate students at The Center for Computational Intelligence) included cclust and fuzzy clustering routines which provide newer clustering algorithms from the machine learning community, strucchange providing a comprehensive toolbox for visualizing of projects, testing and monitoring structural changes in linear regression models, tseries for computations in finance. [8] To compliment the growing community of R users, user control over graphics and its functionality was divided into small packets that could be added or removed when the need arose.

3.2 Changes in R from S

Gentleman and Ihaka added specific data structures that had statistical concepts and decided to select a basic user interface that used a system akin to S. Despite the similarities in S and R, changes in R in the realm of data management and scoping separate the languages. [9] R specifies a set amount of memory at start up and offer dynamic garbage collection. This change was vital to minimize heap growth and paging errors that proved problematic in S.

The changes in scoping throughout R allowed functions access to the variables which were in effect when the function was defined; an idea which dates back to Algol 60 and found in Scheme and other lexically scoped languages. It is one of R's strongest attributes, flexibility in creating functions by allowing objects in functions to remain local to that function and the ability to return any data type. [6] Functions in R return the last object/call that is performed within the function.

4. CRITICISM

There are three typical criticisms levied against R. Some are systemic, rooted in the design and the era in which it was created that are antiquated today, and others are of personal choices and inclinations. The first two are bound by the former arguments and the latter reasoning defines the last criticism. The R-Core concede a new platform needs to be developed stay align R with the ever-changing standards of modern computing.

4.1 The slow nature of analysis

The first criticism of R is rooted in the era in which R was first designed. In the mid-1990s, multi-core process was in an infantile stage comparable, so the core design of R is restricted to the technology of the era. The thought of including architecture that would support multi-processor manipulation was not in the common nomenclature. This has been eased over the years as advanced in that department which make it possible to implement tweaks to the system and corrections made in the initial design. As an example of such advancements, R's computations have become vectorized in an attempt to let multiple processes that can be partitioned and allow subprocesses dedicated to multiple processors that work in tandem as they are forked into new subroutines.

4.2 Inability to process massive data

The second criticism is based around R's "pass by value" system. At the heart of this problem lies the crux of the complaint: systemically, each time a variable is needed it is copied from its source memory location and placed into the function or procedure that needed it. When one has millions upon millions of data points, each point will get copied when it is needed for each and every computation. Objects, as well, are stored in physical memory, so the more elaborate the design of the object and more numerous the occurrences of the object, the higher the demand on the system to supply the computational power to augment and manipulate data. This problem is a fundamental and will be reviewed because it will require rewriting the entirety of the program to convert it to a more modern adaptation.

4.3 Availability and Age

A third criticism is that packages are based on user-demands. R draws most of their advancements in design from inspirations set forth by the community of R users. Packages that reach compliance with R-Core standards are made available to the general user. If proven durable and in high enough demand, eventually the package is added to the base R dataset. However, the problem lies in this accessibility. If there is no demand for a particular methodology, packages have to be written by the intended user. Moreover, as its internal workings are nearly 30 years old, R is not ideal for all situations and can be a arduous ask for extremely specialized packages that, while could be very helpful – even more so beyond the scope of its initial inception – they remain bound to a balance between those with a proclivity to create material for the platform and those that use R on a more moderate basis.

5. OO PROGRAMMING

R's object-oriented structures are divided into two groups: functional classes and encapsulated classes. Functional classes are the original class forms in R which are S3 and S4. The encapsulated classes are denoted with an R and are not as of yet native types in R. The core syntax of base R is as follows in figure 1: [5]

e::= n | s | x | x[[e]] | {e; e}

| function(f) e

| x(a) | x<-e | x<<-e

| x[[e]] <-e | x[[e]]<<-e

| attr(e,e) | attr(e,e)<-e

| u | v(a)

F::= x | x = e

a::= e | x = e

Fig. 1

5.1 Functional OOP

S3 is the initial incarnation of classes and methods in R language. A class is a direct reference to an object with slots being a class-specific field. Methods are functions that carry out computations on the objects. S3, a natural progression from its predecessors, chiefly S1 and S2 from the S language, offers inheritance though a method called NextMethod. This call would create another method, not dependent on the object by the state of the generic methods built into the R system. Methods invoked behave as if they had been invoked from the previous method - arguments to the inherited method are in the same order and have the same names as the call to the current method. This means that they are the same as for the call to the generic methods but the expressions for the arguments are the names of the formal arguments of the current method and have values that correspond to their value at the time NextMethod was invoked. [7] The ease of use with this system of object creation and methodolgy has made S3 a staple in any R programmer since its origination.

S4 worked very similar to S3 but with stricter implementations that was in line with other OO language. From S3 to S3, there were two major differences, beyond the tightening of protocol, which are more expressive formal class definitions, especially concerning inheritance, and the fact that generic functions can dispatched to a method based on a class of containing any number of arguments, not just one. In s3 you could simply turn an object into a class by setting the attributes to mirror that of the respective class, but S4 requires a new method, created using the setClass method which can only be used with a constructor function aptly called new. When constructing a new class object, if any of the slot are left out the newly constructed class it will become a default generic class assigned automatically by R's internal logic.[8] Methods are also regulated with a view to generics in mind. For example, callNextMethod is the method used to ensure the most generic version of a method is used. This is because generic methods are tried and viable when compared to user-created methods which can, when not designed properly, complicate a functional process. S4, depending on the type of programmer asked, is the most powerful, yet misunderstood method in R due to its stricter rules but the flexibility, power, and accuracy in defining objects to specific classes and evaluation and method use on these classes show the potential of the inate design of R.

5.2 Encapsulated OOP

The encapsulated OOP portion of R started with R5 which does not refer to a typical OO class, rather a reference class. Reference classes resemble classes is C and Java languages. Instead of having generic functions as in the case of S3 and S4, reference classes allow for encapsulation of methods based on its definition. [9] R began to privatize their class system with methods directly derived of a given class being the only way to manipulate and extract class data. New methods and classes are created via the SetRefClass call and all functions inside a class denoted by the use of $ preceding the method call.[10] The chance of the object being referenced is changed is solely up to the actions within the method being called, a powerful but dangerous proposition. As such, references classes have yet to make the base edition of R.0

In that vein, R6 is the latest object class type. It creates classes with reference semantics, similar to R's built-in reference classes. Compared to reference classes, R6 classes are simpler and lighter-weight, and they are not built on S4 classes so they do not require the methods package. These classes allow public and private members, and they support inheritance, even when the classes are defined in different packages. That said, R6 is not currently supported inside R's base package.

5.3 Nested Classes

Nested Classes are supported in R. Much like its programming counterparts, for, if, and while loops can be implemented singularly or by nesting one inside another. [11] The same applies to classes. This is achieved by Rs adaptation of lexical scoping. Nesting is achieved by declaring variables and functions into the environment of the parent class (leading to inheritance to child classes) or function method. This is enabled by scoping classes at compile time, places them appropriately in the source code, then execute in the given scope of the class's definition.

6. REFERENCE

[1] “Genesis.” *R*, cran.r-project.org/doc/html/interface98-paper/paper_1.html.

[2] “An Updated History of R.” *Revolutions*, blog.revolutionanalytics.com/2017/10/updated-history-of-r.html..

[3] Level. “How Big Companies Are Using R for Data Analysis.” *Level Blog*, 4 Aug. 2017, www.northeastern.edu/levelblog/2017/05/31/big-companies-using-r-data-analysis/.

[4] Ihaka, Ross. *The R Project: A Brief History and Thoughts About the Future*, University of Auckland, www.stat.auckland.ac.nz/~ihaka/downloads/Massey.pdf.

[5] Morandat, Floreal, et al. “Evaluating the Design of the R Language: Objects University and Functions For Data Analysis.” R-ecoop12.Pdf, Purdue, r.cs.purdue.edu/pub/ecoop12.pdf.

[6] Christian, Nicholas. “Advanced Programming.” Advanced Programming Lectures, University of Pittsburgh, 2011, www.pitt.edu/~njc23/Lecture4.pdf.

[7] “NextMethod.” *R Language Definition*, cran.r-project.org/doc/manuals/r-release/R-lang.html#NextMethod.

[8] Wickham, Hadley. “The S4 Object System.” *S4 · Advanced R.*, adv-r.had.co.nz/S4.html.

[9] “Reference Classes.” *Function | R Documentation*, www.rdocumentation.org/packages/methods/versions/3.4.1/topics/ReferenceClasses.

[10] Chang, Winston. “Encapsulated Classes with Reference Semantics [R Package R6 Version 2.4.0].” *The Comprehensive R Archive Network*, Comprehensive R Archive Network (CRAN), cran.r-project.org/web/packages/R6/index.html.

[11] “Nested Design in R.” *Nested Designs in R*, Pennsylvania State University, www.personal.psu.edu/mar36/stat_461/nested/nested_designs.html.