Interfaces from R and Extensions

(or, Shakespeare Meets R)

John M Chambers

1 Introduction

In the modern software-rich environment, a project with serious computing needs is likely to benefit from using tools in more than one programming language, as long as the interface allows users easy access to the computing they need without a serious learning barrier.

In the following sections, some techniques are explored that may help implementers provide such techniques for application users. We will build on a general approach to interfaces described in [1]. The running example will be a package with a modest goal, the use of some data analytic techniques to explore Shakespeare's plays for possibly interesting patterns. Admittedly not of great importance compared to the major modern challenges, but a subject that has generated much discussion for centuries. Facilities for Interfaces and object-oriented programming in R will benefit the application, as is often the case. We will also emphasize programming to extend these techniques.

1.1 Interfaces and Extensions

Using an interface from one programming language or environment to another is a winning strategy for projects with challenging computing requirements. In the "Extending R" book (@extR, hereafter *ExR*) this strategy is related to R, as one of the three fundamental principles: "Interfaces are part of R".

When essential programming idioms are shared between the languages, the use of an interface can be made natural for programming by defining *proxies*. In particular, both R and many other powerful languages emphasize versions of *functions* and *classes*.

An interface from R to such a language can define proxy functions and proxy classes, to be used in a natural R way but which in fact are automatically interfaced to corresponding program constructs in the other language. In addition to using such proxies straight-out, the application is likely to extend the computations and incorporate them in functions, classes or other structures in R.

We'll look at some techniques for such extensions, particularly for embedding proxy classes in R classes. For a more-or-less realistic project in which these might be useful, we'll consider a package to apply data analysis to the plays of William Shakespeare.

1.2 Shakespeare Meets R, and XML, and Python

For over four centuries, Shakespeare's plays have been the focus of a sea of studies and debates, including some recent work applying statistical and natural-language techniques, focussed on questions of authorship in particular.

In a more exploratory spirit, there are a number of interesting questions for which R is suited (well, interesting to me, certainly). How do the plays change with time, or with subject matter? Are characters distinguished in speech by class, by gender or by other context?

Suppose we start on a project to construct tools in R to explore such questions. First question: the data. The plays have been digitized, in some different forms. For our purposes, one of these has the key advantage of preserving the most structural information in a way that can be useful for analysis. Thirty-seven plays were transcribed in HTML format and later converted to XML.

The great asset of XML is that it provides a hierarchical grammar for describing objects in a tree-like structure, using specific terms for the particular objects. For the plays, this organizes the data by acts and scenes. Within each scene speeches, stage directions and a few other items are explicitly distinguished. All this structure is available to us for analysis.

For example, the most important part of the plays will be the speeches, those "words, words, words" as Hamlet says. In XML parlance, a speech has a corresponding tag and structure:

```
<SPEECH>
<SPEAKER>OBERON</SPEAKER>
<LINE>Ill met by moonlight, proud Titania.</LINE>
</SPEECH>
```

References

References

[1] John M. Chambers. *Extending R.* Chapman & Hall, 2016.