Bioconductor Project
Bioconductor Project Working Papers

Year 2004 *Paper* 2

Statistical Analyses and Reproducible Research

Robert Gentleman* Duncan Temple Lang†

*Department of Biostatistics, Harvard University, rgentlem@fhcrc.org
†Department of Statistics, University of California, Davis, duncan@wald.ucdavis.edu

This working paper is hosted by The Berkeley Electronic Press (bepress) and may not be commercially reproduced without the permission of the copyright holder.

http://biostats.bepress.com/bioconductor/paper2

Copyright ©2004 by the authors.

Statistical Analyses and Reproducible Research

Robert Gentleman and Duncan Temple Lang

Abstract

For various reasons, it is important, if not essential, to integrate the computations and code used in data analyses, methodological descriptions, simulations, etc. with the documents that describe and rely on them. This integration allows readers to both verify and adapt the statements in the documents. Authors can easily reproduce them in the future, and they can present the document's contents in a different medium, e.g. with interactive controls. This paper describes a software framework for authoring and distributing these integrated, dynamic documents that contain text, code, data, and any auxiliary content needed to recreate the computations. The documents are dynamic in that the contents, including figures, tables, etc., can be recalculated each time a view of the document is generated. Our model treats a dynamic document as a master or "source" document from which one can generate different *views* in the form of traditional, derived documents for different audiences.

We introduce the concept of a *compendium* as both a container for the different elements that make up the document and its computations (i.e. text, code, data, ...), and as a means for distributing, managing and updating the collection.

The step from disseminating analyses via a compendium to *reproducible research* is a small one. By reproducible research, we mean research papers with accompanying software tools that allow the reader to directly reproduce the results and employ the methods that are presented in the research paper. Some of the issues involved in paradigms for the production, distribution and use of such reproducible research are discussed.

Statistical Analyses and Reproducible Research

R. Gentleman
D. Temple Lang

May 29, 2004

Abstract

For various reasons, it is important, if not essential, to integrate the computations and code used in data analyses, methodological descriptions, simulations, etc. with the documents that describe and rely on them. This integration allows readers to both verify and adapt the statements in the documents. Authors can easily reproduce them in the future, and they can present the document's contents in a different medium, e.g. with interactive controls. This paper describes a software framework for authoring and distributing these integrated, dynamic documents that contain text, code, data, and any auxiliary content needed to recreate the computations. The documents are dynamic in that the contents, including figures, tables, etc., can be recalculated each time a view of the document is generated. Our model treats a dynamic document as a master or "source" document from which one can generate different *views* in the form of traditional, derived documents for different audiences.

We introduce the concept of a *compendium* as both a container for the different elements that make up the document and its computations (i.e. text, code, data, ...), and as a means for distributing, managing and updating the collection.

The step from disseminating analyses via a compendium to *reproducible research* is a small one. By reproducible research, we mean research papers with accompanying software tools that allow the reader to directly reproduce the results and employ the methods that are presented in the research paper. Some of the issues involved in paradigms for the production, distribution and use of such reproducible research are discussed.

Key Words: Compendium, Dynamic documents, Literate programming, Markup language, Perl, Python, R.

COBRA
A BEPRESS REPOSITORY
Collection of Biostatistics
Research Archive

Hosted by The Berkeley Electronic Press

1 Introduction

Statistical methodology generally involves algorithmic concepts. The descriptions of how to perform a specific analysis for a given dataset or generally how to perform a type of analysis tend to be similarly procedural or algorithmic. Expressing these concepts in a purely textual format (such as a paper or a book) is seldom entirely satisfactory for either the author or the readers. The former is trapped in a language that is not conducive to succinct, exact expression and the audience is separated from the actions and details of the algorithm and often forced to make assumptions about the precise computational details.

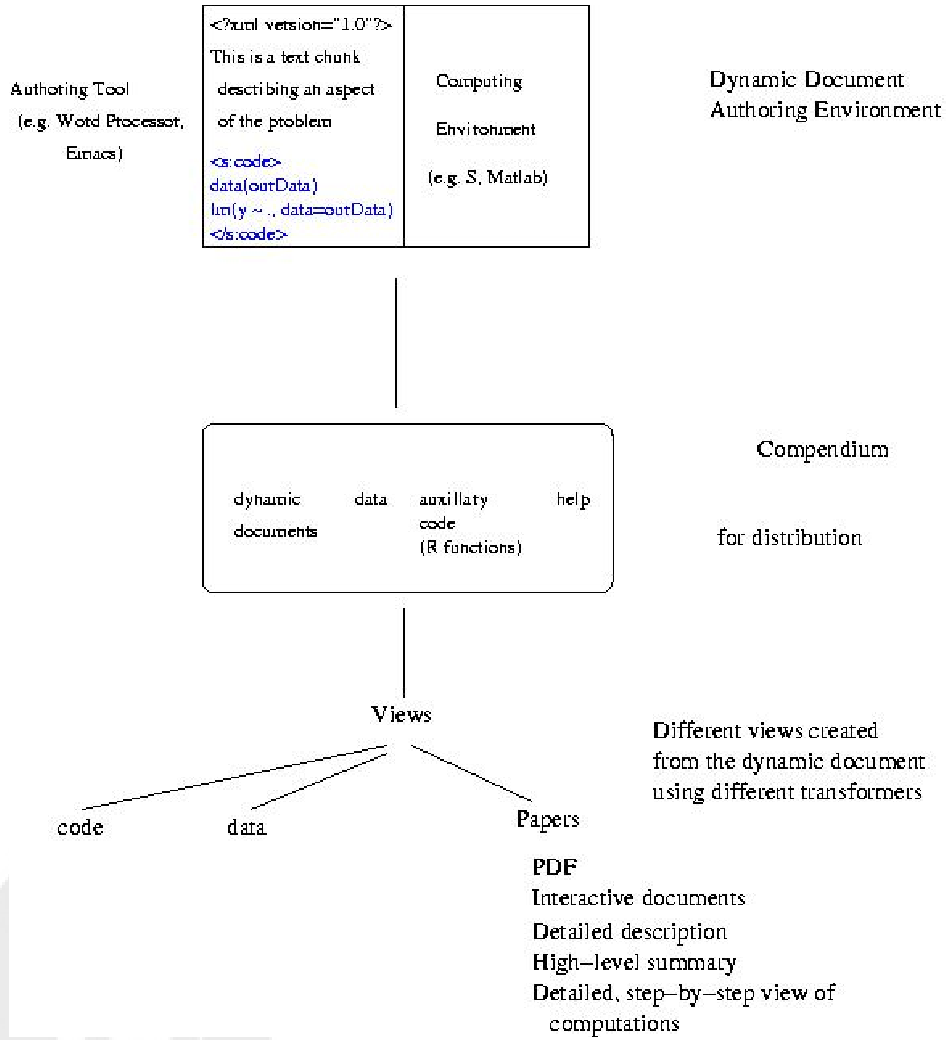
Some of these difficulties can be overcome by supplying computer code and data that explicitly describe the required operations. While there are many examples of such approaches, there is no generally good strategy that benefits both authors and readers alike. There are few instances which do not simply relegate the associated code and data in a disjoint manner to appendices or which use references to a Web site, thus leaving each user to navigate between the text, data and code herself. She must also manually synchronize different versions of the different inputs. In this article, we describe a new mechanism that combines text, data, and auxiliary software into a distributable and executable unit which we will refer to as a *compendium*.

A compendium contains one or more *dynamic documents* which can be transformed into a traditional, static document in much the same way we generate PDF (Portable Document Format) files from TEX. However, what is important about the concept of a compendium is that the dynamic documents can also be transformed and used in other ways. The elements of a dynamic document (i.e. text, code, data) can be extracted and processed in various different ways by both the author and the reader. Hence a compendium has one, or more, self-contained *live* documents that can be regenerated in absolute detail by others, and that can often be used in contexts other than the author's original work. Figure 1 presents the different pieces and the flow from the creation of the dynamic documents to the creation of different views.

Since we will use common terms in specific ways throughout the paper we begin by defining and describing them. We will use the terms *reader* and *user*

COBRA
A BEPRESS REPOSITORY
Collection of Biostatistics
Research Archive

http://biostats.bepress.com/bioconductor/paper2



COBRA
A BEPRESS REPOSITORY
Collection of Biostatistics
Research Archive

Hosted by The Berkeley Electronic Press

interchangeably; both convey the notion of a member of the audience for which the compendium was produced by the author. We emphasize at the outset that the mode of interaction with a "published" compendium is determined by the user not by the author. We define a *dynamic document* as an ordered composition of code chunks and text chunks that describe and discuss a problem and its solution. The ordering of the chunks need not be simply sequential but can support rich and complex document structures. For example, the ordering of the chunks may have branches and generally may form a graph with various paths through the nodes (chunks) that allow different readers to navigate the document in different ways.

*Code chunk*s are sequences of commands in some programming language such as R or Perl. Code chunks are intended to be evaluated according to the language in which they are written. These perform the computations needed to produce the appropriate output within the paper, and also to produce intermediate results used across different code chunks. *Text chunk*s describe the problem, the code, the results and often their interpretation. Text chunks are intended to be formatted for reading.

By *auxiliary software* we mean software that is specific to the problem at hand but which does not appear in the dynamic document. Auxiliary software is not general purpose software such as R, SAS or Perl or any general libraries, but rather it is support utilities that simplify the computations in the code chunks in the particular dynamic document (or family of related documents). It might be R functions, Perl subroutines, or C routines. For example, such auxiliary code might fit a particular statistical model and the dynamic document may include calls to the model fitting routine but the document would not necessarily contain the model fitting code. Each separate language (or system) used in a dynamic document may have its own auxiliary software. Essentially, auxiliary software allows us to more effectively organize the computations in code chunks in one or more dynamic documents. By allowing support software to exist outside of the dynamic documents, we avoid redundant copies of the same code, and allow the functionality to be used independently of the dynamic documents, for example by readers who want to use the basic computations described in the dynamic documents in different ways.

The mechanism or system used to transform a dynamic document into some desired output or view will be called a *transformer*. The transformer takes as input

COBRA
A BEPRESS REPOSITORY
Collection of Biostatistics
Research Archive

http://biostats.bepress.com/bioconductor/paper2

a compendium and the desired target and produces the appropriate output. Thus, it is responsible for identifying the appropriate languages for all code chunks, for ensuring access to any auxiliary software, for evaluating the code chunks in the appropriate environment, for assembling the outputs and finally for producing the desired result. Transformers can be written in any high level language.

Generally speaking a dynamic document will describe a problem such as a proposed methodology, a data analysis, a simulation, or a tutorial on a particular topic. The compendium is a mechanism for associating both the data and the software needed to process the data, together with the text that the author wants to present. A dynamic document provides a means for interweaving the textual description of the problem and the computer code that processes the data to produce the necessary facts, figures and tables that the author wants to present. Finally, the transformer provides the means for turning the dynamic document into different desired outputs such as static papers or web pages. Importantly, it is the compendium that is distributed so that the data and auxiliary software are available to the reader. This allows readers themselves to generate and interact with different views.

A reader may have many different interactions with a single compendium. For example, a user might extract only the code chunks from a dynamic document, or perhaps only those code chunks for a particular language. Alternatively, a user might create a traditional document which contains the text intermingled with various outputs derived from the code chunks. In this paradigm, tables and figures in the output documents are not susceptible to the common errors introduced by manually inserting them and they are automatically updated, by reprocessing, when earlier computations or inputs are modified. This mixing together of computer programs and textual descriptions that can be transformed into different views is the basis of literate programming (Knuth, 1992). In this paper, we consider the adoption and adaptation of some of the ideas put forward by Knuth and others.

Essentially, we are describing an executable or "*runnable*" document that can be used to recreate the author's original text and results. However, dynamic documents also provide the reader with the explicit details of all computations. Thus compendiums are a specific form of software and extend our usual notions of documents. This concept of a dynamic document can itself be extended to allow the reader to parameterize the computations with different inputs such as tuning pa-

COBRA
A BEPRESS REPOSITORY
Collection of Biostatistics
Research Archive

Hosted by The Berkeley Electronic Press

rameters for algorithms or alternative datasets. And a different view derived from the compendium facilitates the readers interactive control of the computations and content as they browse that view/document.

1.1 Applications

An increasingly important aspect of statistical practice is the dissemination of statistical methodology, data analyses and statistical reasoning. While statistical practice has evolved to encompass more computation and larger and more complex datasets and models, the primary vehicle for delivery has remained the static, printed page. We believe that the concepts of dynamic documents and compendiums will greatly facilitate disseminating sophisticated statistical concepts and analyses. They allow the author to express her ideas using a combination of languages, each appropriate to the particular concept. They allow the reader to explore both the document and the entire compendium in her own way. This allows her to navigate the details of the actual computations as desired in a non-linear manner. We are describing a framework that would allow, for example, the reader, upon encountering a figure in a paper to obtain explicit details about the computations that generated that figure. Ideally, they would have access to the inputs and be able to explore alternative computations or parameter values. Compendiums, combined with the tools we envisage, provide this and many more interactive opportunities for the reader which engender fundamental changes in the way we read and use technical documents.

Compendiums have the potential to be useful in a broad array of disciplines and activities. In the interest of concreteness, we consider two related situations where the sharing of data, scripts for analyzing the data, and a discussion of the analysis are important. The first example is the conveyance of a particular analysis which has scientific merit but which is quite complex. Our second example deals with tutorials on good statistical practice that are intended for students or practitioners from other disciplines. These situations have their roots in pedagogy, reproducible research and information science. The focus is not on statistics itself, but on the dissemination of descriptions of statistical analyses. Issues that will need to be considered include obtaining, interacting with and curating compendiums.

In the first case there is a particular analysis (methodology and data) that the

COBRA
A BEPRESS REPOSITORY
Collection of Biostatistics
Research Archive

http://biostats.bepress.com/bioconductor/paper2

author wants to convey. In the field of computational biology, complex analyses are routinely applied to large, intricate data sets. Page limits on printed material prevent complete descriptions of the analyses to say nothing of the problems, mentioned earlier, of conveying algorithmic concepts in English. To overcome this problem, many authors provide the data and code as separate, supplemental materials. This also tends not to be completely satisfactory since each author chooses different conventions and typically has no good way of conveying the exact set of steps involved.

In concrete terms, the author selects the data that she will use to defend a particular point of view or conclusion. She then transforms that data to produce figures, tables and to fit models. The output of these computations is assembled into the finished document and used to convince the readership that the point of view or conclusion is valid. In these terms one may then view papers based on computation as advertisement. There is a leap of faith required by the reader; they must believe that the transformations and model fitting were done appropriately and without error.

The compendium concept can alter this situation quite dramatically. The original printed material can be created directly, by author and reader alike, from a compendium using one transformation. But, additionally, the code and data will allow the reader to exactly replicate the reported analysis. The inclusion of the entire history of computations performed by the author allows readers to examine the techniques at different levels of detail. Implicit assumptions and mistakes can be discovered by peer review in the scientific tradition. Publication of more than the final results should be the rule rather than the exception. And while compendiums allow curious readers to *zoom in* on details, they also permit more casual readers to view just the higher level content. Additionally, different transformations can create views that exploit different media, such as for interactive viewing and presentations.

In the second instance, we consider a situation where an individual would like to provide a detailed and explicit description of how to use a particular statistical method. For example, we might like to explain cross–validation as a general principle while possibly providing some more elaborate extensions. Such documents are routinely produced (though not always published) by individuals and research

COBRA
A BEPRESS REPOSITORY
Collection of Biostatistics
Research Archive

Hosted by The Berkeley Electronic Press

groups. In most cases, their authors have produced accompanying software, but in the absence of a standard mechanism for distribution, the software is often overlooked or treated separately from the document by both author and reader. And generally different authors use different solutions, thereby imposing a substantial burden on users who move between the text and the tools being described.

In both examples considered above, the author would like to provide code, data, and textual information in a coherent framework. The compendium provides a method to combine and distribute these materials. Much of the compendium concept is merely the adoption of specific conventions, and can, in some sense, be considered a form of markup or "packaging".

We have discussed some applications that will be immediately available and interesting to authors and readers alike. It is also useful to summarize some of the many uses and implications of compendiums:

- they encapsulate the actual work of the author, not just an abridged version, and allow different levels of detail to be displayed in different derived documents;
- they are easy to re-run by any interested reader, potentially with different inputs;
- by providing explicit computational details they make it easier for others to adapt and extend the reported methodology;
- they enable the programmatic construction of plots and tables (in contrast with most of the current methods that are equivalent to cut and paste methodologies and have the associated problems);
- they allow the document to be treated as data or inputs to software and manipulated programmatically in numerous ways.

The compendium framework lends itself immediately to the concept of *reproducible research* as described by Buckheit and Donoho (1995). They proposed that any published figures should be accompanied by the complete software environment necessary for generating those figures. Extending these ideas to a more general context where the content and results (i.e. tables, figures, etc.) being published rely on computation or on data analysis suggests that the authors should

COBRA
A BEPRESS REPOSITORY
Collection of Biostatistics
Research Archive

provide explicit inputs and code that can be used to replicate those results. The compendium concept provides the necessary structure to satisfy these demands and, in fact, is capable of supporting a much richer set of ideas and operations than those in Buckheit and Donoho's original proposal.

The layout of the paper is as follows. In the next sections, we discuss concrete aspects of our proposal, beginning with motivation and followed by a framework for software that implements the concepts. Finally we comment on an implementation, or prototype, that is currently available. In Section 6, we consider future directions and extensions of the concepts put forward in Sections 1 through 5. This is followed by a discussion of the concepts and some of the more general implications. Because there are a number of specialized terms and software components that may not be familiar to all readers, we have provided a glossary. The specific details of different languages and their Web locations and other details are confined to the glossary. We are intentionally minimizing our focus on the actual implementation of such a system. Our purpose is to promote the benefits of, and the need for, viewing documents involving computations in broader terms that include the computations and data themselves and not just their static output or representation. We also consider the software infrastructure that will be needed to support a general version of these documents.

2 Concepts in Dynamic Documents

Before focusing on a prototype system for creating and processing compendiums, it is beneficial to consider some important general concepts that form the foundation and motivation of our approach.

2.1 Literate Programming

A few of the main concepts of literate programming are well known, but have not been widely adopted. As noted previously one of the major requirements of reproducible research is to provide methodology that allows the author to easily assemble and relate both textual and algorithmic descriptions of the task or research being described. In many cases algorithmic descriptions are more useful to the reader if they are in the form of (annotated) runnable code.

COBRA
A BEPRESS REPOSITORY
Collection of Biostatistics
Research Archive

Hosted by The Berkeley Electronic Press

Literate programming is an idea that was introduced by Knuth (Knuth, 1992) and implemented in a variety of software tools such as `noweb` (Ramsey, 1994). A literate program is a document that is a mixture of code segments and text segments. It is written to be read by humans rather than a computer and is organized as such. The text segments provide descriptions and details of what the code is supposed to do. The code itself must be syntactically correct but need not be organized in a fashion that can be directly compiled or evaluated.

A literate program should support two types of transformation: *weaving* and *tangling*. A program is tangled to suppress the textual content and arrange the code segments into a form suitable for machine processing (such as compilation or direct evaluation). In short, one tangles a document to get usable code. A program is woven to produce a document suitable for human viewing. In its standard sense, weaving produces a document that displays the code and its annotations. In our world of dynamic documents, we use the term weaving to describe the process of creating the document for the reader, with content generated during the transformation to create the figures, tables, etc. by executing, or evaluating, the code and inserting its output into the document. In this sense, the content of the document is dynamically generated. The essential idea we are borrowing from literate programming is the combination of the text and code within the same document. An additional consequence is that we can leverage the structure of the compendium to provide programmatic processing of the different elements of the documents to provide richer views, essentially treating the compendium as data itself.

While literate programming has never gained a large following, it seems that there are some good reasons to recommend it to statisticians. The original intention of literate programming was to provide a mechanism for describing a program or algorithm, but it may be more useful as a mechanism for describing data analyses and methodologies (either explicit or conceptual). When coupled with other tools for testing and validating code, it provides a powerful mechanism for conveying descriptions, carrying out reproducible research and enhancing readability and understanding.

Temple Lang (2001) describes an approach for processing dynamic and interactive literate documents using R and XML. *Sweave* (Leisch, 2002) provides an environment within the R system and more traditional tools (e.g. LaTeX) that al-

COBRA
A BEPRESS REPOSITORY
Collection of Biostatistics
Research Archive

http://biostats.bepress.com/bioconductor/paper2

lows us to mix a narrative (textual) description of the analysis together with the appropriate code segments. An Sweave document is similar to a literate program (apart from minor technical differences) and is basically a mixture of code and text. The text is marked up in a LaTeX like syntax. When weaving the document, there is a great deal of control that can be exhibited over the running of the code segments and the interested reader is referred to the documentation for Sweave. The end result of weaving is a LaTeX file that can be further processed into PDF or any other desired format. The result of tangling an Sweave document is the code – extracted and rearranged – that can be used within R.

There has been some interest in literate programming and reproducible research within the statistical community for some time now. Examples include, Carey (2001) and Buckheit and Donoho (1995). Rossini (2001) provides an overview of the area. Sweave uses a traditional literate programming format to mark up the text and code chunks of a dynamic document. Other markups have been used for the same effect. XML is a natural choice with many significant and useful benefits and advantages.

2.2 Reproducible Research

Recall that by reproducible research, we simply mean that individuals in addition to the author can recreate all of the computations described in a body of work. Since a compendium contains *all* of the inputs (e.g. data, parameters) for the computations in every dynamic document it contains, it is independently reproducible. Readers can transform each dynamic document in the compendium in the same manner as the author to generate the author's published view. The author can use this reproducibility to verify and update her original report by running it on a different machine or at a different time. And when delivered to another reader, the compendium constitutes independent reproducibility; the author has provided sufficient detail (in the form of code and data) for a reader to reproduce the details of the author's presentation. The user can simply invoke the programs that the author has provided and verify that (using the data and code the author has provided) one can create the tables, figures and other outputs on which the author has based her conclusions. Note that it is important to separate the idea of reproducible research from independent verification or scientific reproducibility. Independent verifica-

COBRA
A BEPRESS REPOSITORY
Collection of Biostatistics
Research Archive

Hosted by The Berkeley Electronic Press

tion requires that others repeat the entire experiment, under similar conditions, and obtain similar results. Compendiums and dynamic documents are useful and necessary for evaluating and verifying the evidence provided by an author, but do not necessarily verify the conclusions or inferences about the subject matter.

Scientific reproducibility requires an independent verification of a particular fact or observation. For *in silico* experiments, on the other hand, this requirement is for an independent implementation of the experiment. The level to which independence in computational statistical research is required is not yet well established. There are many different levels possible and while the *de facto* standard seems to be an independent implementation in some high level language, legitimate concerns still exist. For example, should one require a different language or is it sufficient to use a different compiler but the same language? Does one have to consider a different operating system? Do all libraries used have to be reimplemented? Clearly bugs in, or failures of, any of these components will affect all experiments that relied on them.

The practical notion of complete repeatability is limited. However, developing a truly independent experiment will generally be easier if reproducible research, in the spirit mentioned here, is available. Compendiums will provide substantially more detail about the process that was actually used to produce the results than a static paper. Compendiums can provide scientific reproducibility but they are not sufficient for independent verification. While the challenges of independent verification are interesting and important, we will ignore them and remain focused on our narrow definition of reproducible research.

It is also important to consider some limitations associated with data capture in the process of creating a compendium. While ideally we would like to capture the data at as early a stage as possible, it is not feasible to do so in a entirely reproducible way (except for simulation experiments). For example, transformations to anonymize the data for privacy reasons may introduce errors but we cannot verify these. Thus, in the context of real data, some decision will need to be made about the point at which the data are captured in the compendium. The compendium then documents all transformations and manipulations from that point forward, but clearly can provide no verification of any previous aspects of the analysis. This is no different than the present situation. Authors that put their data on the Web must

COBRA
A BEPRESS REPOSITORY
Collection of Biostatistics
Research Archive

http://biostats.bepress.com/bioconductor/paper2

make some decision with regard to what their starting point is. However, the compendium approach has the advantage of ensuring that the supplied data are capable of reproducing the claimed outputs. Baggerly et al. (2004) demonstrate many of the problems that can arise when using standard publications together with author supplied data. After having expended considerable effort they are unable to reproduce the results claimed. We also note that there is nothing in the concept of a compendium that prohibits the capture of more details. It is a practical matter, not a conceptual one.

While we have described weaving and tangling as two transformations to create different views of a dynamic document and compendium, many other transformations are possible. At its simplest, we can include or exclude different text chunks for different audiences, for example to present more or less detail, or different aspects of the analyses. Generally, adding more detail to the compendium is valuable, since it need not be displayed in the documents generated for readers, but is available to interested parties. We would also like to point out that there is, in principle, no need to put all aspects of an analysis into a single document or compendium. A compendium can contain multiple dynamic documents, and the early manipulations could be contained in one document while the analyses are described in another. Similarly, an author might separate these into different compendiums, one suitable for publication and the other for reference. Users that want to understand the whole process would obtain two compendiums while those that wanted only to understand one of the aspects would obtain only the appropriate compendium for their interests. Since the dynamic documents are programmatically accessible and the transformers can operate on different aspects of different documents, the choice of describing different aspects of a study in a single document or several documents can be left to the author and is a matter of style and convenience.

2.3 Conceptual Overview

It is worth briefly reviewing and adding some context to the ideas that have been expressed. A compendium is a collection or archive containing data, code and text. A compendium can be processed in many different ways to produce many different outputs or *views*. The author makes certain transformations available (e.g. a PDF file, a script file containing the commands to perform all the computations) and

COBRA
A BEPRESS REPOSITORY
Collection of Biostatistics
Research Archive

Hosted by The Berkeley Electronic Press

the user/reader has complete access to the computational details and can apply or carry out any of the transformations on her own computer at any time. There are basically two different types of processing or evaluation. One is the transformation of the data inputs by the code chunks via one or more specific programming languages (e.g Perl or R) to provide output. A second type of processing takes the outputs from these evaluations, combines them with textual descriptions (in many cases) and provides a narrative output for the user. In the abstract there is no clear delineation between these two steps as all transformations are merely computations on "data" – problem-related or document elements – to yield outputs. This is unfortunate from the perspective of providing a simple, stepwise explanation of how the creation of a document works, but it greatly reduces the complexity of and enhances the processing itself.

In simple terms, processing a compendium consists of two sets of computations. One set pertains to processing the structure of the compendium by identifying and manipulating the different text, code and data elements. The second set of computations involves evaluating the code chunks within the compendium structure. The evaluation of each code chunk will take place in the appropriate language for that code and is delegated to that programming system. So this set of computations for the code chunks may involve one or more different programming languages such as R, Perl or Matlab. The first set computations on the structure of the compendium can be written in any general programming language. What is imperative is that some form of markup language is needed to identify different components of a dynamic document.

3 A Functional Prototype

In the interest of concreteness, we describe software tools that are currently available with which we constructed a basic, functional compendium. We note that the concept of a compendium is much broader and that there are already more general implementations than we will detail in this section. We present this as a "strawman" so that we can discuss the need for a more general system.

Our goal is to start with the data in some raw or unprocessed state and describe the set of transformations or computations, jointly in words and in software, that

COBRA
A BEPRESS REPOSITORY
Collection of Biostatistics
Research Archive

http://biostats.bepress.com/bioconductor/paper2

are needed in order to properly interpret the data. The resulting compendium can be processed by the author, or by any reader, to yield a final report. After describing the general structural requirements, we provide some explicit details about Sweave and its role in our prototype.

This prototype can be thought of as a single-language compendium. That is, we will make heavy use of a single language – R (Ihaka and Gentleman, 1996), in order to create, distribute and transform the compendiums. The code chunks will be restricted, being written only in this single language. This is not such a restriction since R has conventions for calling code written in other languages, but the author cannot detail the use of different tools and languages he may use in practice. This is an issue in a general system, but not for our description of the basic architecture and authoring pipeline.

To create compendiums, we need a way to combine data, problem specific software and scripts together with the text chunks in such a way that the desired computations can be carried out. And we must provide access to the more general data analytic and computational tools needed to carry out the analysis, e.g. R. We will also consider the need for tools to verify (test), maintain and distribute compendiums. And readers of compendiums will need tools for locating, obtaining and installing them.

The major components in the prototype are the R package system (R Development Core Team, 1999), Sweave, and the R programming environment itself. The existing package system in R is able to satisfy most of the requirements of combining, verifying, maintaining and distributing the elements of a compendium. This system insists on a particular hierarchical structure for the files that make up a package and provides software to help create, distribute and install such packages. Complete details can be found in the documentation for R (R Development Core Team, 1999). For this discussion, it is sufficient to mention that there are different locations (i.e. directories or folders) in which to place R functions (`R/`), datasets (`data/`), C/C++ and Fortran code (`src/`), help files for the R code (`man/`), and documents (`inst/doc/`).

The dynamic documents are written in a modified version of the `noweb` markup, with text chunks written in a modified version of LaTeX and the code chunks written as regular R code. Sweave provides software for processing dynamic documents to

COBRA
A BEPRESS REPOSITORY
Collection of Biostatistics
Research Archive

Hosted by The Berkeley Electronic Press

create different outputs. Primarily the transformations are either weaving, to obtain a finished document or tangling to extract the code chunks. Thus, Sweave plays the role of the transformer. It takes a dynamic document and transforms it (mainly using tools in R) into the different outputs. A reader can use Sweave, or any other available tools, to carry out desired transformations at any time.

Representing a compendium as an R package has many benefits. The author has a convenient and structured way of organizing each of the data, the auxiliary software, scripts, and the dynamic document. There are a variety of testing and verification tools available in R that allow the author to process the document and to compare various outputs with those obtained on previous runs. The `man/` directory allows (and encourages) the author to document any auxiliary functions. Importantly, the data and auxiliary functions can be used independently of the dynamic documents providing additional utility of the compendium. However, it is important that the reader not confuse a compendium with an R software package: the former is intended to provide support for a scientific paper while the latter is a general piece of software that can be applied to a variety of inputs.

Some examples of compendiums written using this paradigm are available through the Bioconductor Project `www.bioconductor.org`. Readers are encouraged to download one or more of these to see specific examples of some of the concepts presented here. In particular Gentleman (2004) is based on the `GolubRR` compendium and provides a more substantial discussion of how to create R based compendiums.

4 General Software Architecture for Compendiums

We now turn our attention to the general set of characteristics we feel are necessary for any implementation of compendiums. In order to see what support is required, it helps to consider what constitutes a compendium and how it might be used throughout its lifetime. A compendium contains one or more dynamic documents. By its nature, a dynamic document requires supporting software for generating the dynamic content. We distinguish between general purpose software and auxiliary software. By *general purpose software*, we mean the basic language interpreters (e.g. S, Perl, SAS, Java, Excel), compilers (e.g. gcc, Visual Basic)

COBRA
A BEPRESS REPOSITORY
Collection of Biostatistics
Research Archive

http://biostats.bepress.com/bioconductor/paper2

and add-on modules such as those available from software archives such as CPAN (Comprehensive Perl Archive Network) and CRAN (Comprehensive R Archive Network). We presume that the necessary general purpose software is available in the reader's environment and that the transformer is able to detect and use it.

The *auxiliary software* is formally part of the compendium, and typically is made up of software specially written for the topic(s) discussed in the dynamic documents in the compendium. For example, a function that embodies code that is used in several places within the code chunks of the dynamic document(s) would be best centralized in the auxiliary software. It could also contain specific versions of any of the general purpose software, such as the particular version of an R package if general availability of that version is unlikely, now or in the future, or the computations depend explicitly on that version. In simple cases the auxiliary software may be entirely included within the code chunks of the dynamic documents.

A compendium can be represented entirely in a suitably marked up document, e.g. an XML document can contain `<data>` `<auxiliaryCode>`, etc. sections.

The software environment for working with compendiums requires relatively few tools. We next list these and subsequently discuss the first three in more detail.

1. Authoring Software:
   tools to enable the author to integrate code and textual descriptions in a narrative fashion to create the dynamic documents;

2. Auxiliary Software:
   a mechanism for organizing the supporting or auxiliary software and data such as C code, S functions, documentation and datasets so that they can be combined with the dynamic document;

3. Transformation Software:
   tools for processing the compendium to yield different outputs, typically involving transformations of the dynamic document(s);

4. Quality Control Software:
   tools for testing and validating a compendium, for both the author and the reader;

COBRA
A BEPRESS REPOSITORY
Collection of Biostatistics
Research Archive

Hosted by The Berkeley Electronic Press

5. Distribution Software:
   tools for distributing the compendiums and for managing them on both the client- (i.e. reader-) side and the server side; on the server side this includes organization and versioning; and on the client side, it includes tools to access the documentation, code and data.

Authoring Software For the compendium to become an accepted publication mechanism, we will ultimately need easy to use tools for creating compendiums and for authoring dynamic documents. Easy integration and editing of code together with the text will be vital. The author should be able to use a text editor or word processor of her choice, but of course, this will depend on demand and whether open source or commercial offerings are available. When editing the text chunks, all the usual tools (e.g. spell check, outline mode) should be available. For writing code, we want the usual tools for that process (e.g. syntax highlighting, parentheses matching) to also be available. The code chunks need to be functional and simple mechanisms for evaluating them while authoring in the appropriate language are essential. Systems that display some of this functionality include Emacs with Emacs Speaks Statistics (Rossini et al., 2004), and AbiWord, the Gnome project's word processing application. Paradigms for leveraging these familiar tools in the context of creating compendiums need to be explored.

Auxiliary Software The components in a compendium (documents, data, auxiliary software) will essentially be arranged by some convention and in such a way that the transformer can locate them. This convention is usually hierarchical in manner (e.g. the R packaged directory structure) where the conceptual units of the compendium map to physical units such as directories/folders and files. Frameworks that provide similar functionality include file archives (e.g. zip and `tar` files), software language package mechanisms (e.g. the R package system and Perl modules), and XML documents which contain not only the contents of the dynamic documents, but also code, data, etc.. Thus, it is perfectly reasonable (and a likely eventuality) that there will be many different forms of compendiums. For some authors, compendiums will be R packages, while for others they will be single XML documents. Both forms have advantages and disadvantages over the other and both can easily co-exist, together with many other additional formats. The essential characteristic of the compendium is that it is an archive or collection which

COBRA
A BEPRESS REPOSITORY
Collection of Biostatistics
Research Archive

http://biostats.bepress.com/bioconductor/paper2

can be accessed programmatically to locate the different components.

Transformation Software We use the compendium simply as a container for elements from which we generate various different views or outputs such as a PDF document, code, or graphics. We need a collection of filters to generate the different outputs or *views*. For example, it would be natural to use a general transformation utility such as XSL to transform dynamic documents that are marked-up in XML. Sweave provides a transforming mechanism for R, and similar filters can be written within and for other languages and communities. Ideally, additional filters, beyond weave and tangle, will be created by users to generate new views (e.g. interactive documents, bibliographies, citations, data relationships) and programmatically operate on the compendium's content.

One might argue that the compendium concept could be simplified. Indeed, many people have produced their own approaches to creating documents that they can produce by running a script. One can easily write an S script to produce all figures in a paper. Similarly, a script that generates LaTeX tables in separate files can be used for the non-graphical content. The use of the `make` facility suggested in Buckheit and Donoho (1995) is an improvement over this as it provides explicit dependencies between computations and content. Any of these somewhat ad hoc approaches is useful and aids others in reproducing the results. The important idea is that the creation of the document is an atomic action in which the inputs are synchronized. We argue that providing a well-defined structure that others can use directly makes it easier for both authors and readers to work with such documents. So, for example, the additional complexity imposed by the R packaging mechanism is very small relative to the gains in familiarity and tools to create, distribute and process the resulting compendiums. Similar gains will be made by adhering to language specific standards for compendium systems developed in Perl or Python, or using standard transformation utilities (e.g. XSL) rather than home grown, community-specific tools. And, importantly, the structure of the compendiums will make them accessible to programmatic manipulation such as refined search, cataloging, versioning, and so on.

One can also argue that the complexity of dynamic documents and intermingling code and text is not generally needed. Instead, if authors provide a script that generates the figures and tables that are then incorporated into the static doc-

COBRA
A BEPRESS REPOSITORY
Collection of Biostatistics
Research Archive

Hosted by The Berkeley Electronic Press

ument then the goals of the compendium will be achieved. There is some truth to that statement and, to a large extent, this is only a minor detail about how dynamic documents are authored. However, there is also great benefit to the single document approach that combines code and text chunks together. By interleaving the code chunks directly into the document and atomically processing the document as a whole via the transformer, we have a well-defined computational model. The dependencies between the chunks are clear and centralized in a single place. From the reader's perspective, the dynamic document removes any questions about which computations were applied to produce a particular estimate, table or figure. The output, the code and the data are inextricably mixed and the reader can determine exactly which processes were applied, in what order and which specific values were used. And there are less connections to manage between the creation of the sub-elements (e.g. figures and tables) and the different code chunks.

A well documented script will provide the same level of detail as a dynamic document. However, there is a disconnect between the processing and the output. The reader must match these different sources together to reproduce the actual computations and output. For the author, using scripts to create the content requires some form of synchronization. A formal mechanism (e.g. the `make` utility Oram and Talbott (1991)) is needed to ensure that this synchronization is accurately done, and this amounts to a need for robust software and careful attention to detail every time the script is run. The construction of a compendium moves that effort and attention to detail into the construction process, and once created the compendium can be processed repeatedly by many users in an identical manner. Many of us have managed this synchronization manually by explicitly cutting and pasting material from the computational environment into the document. At best, this is tedious and must be undone and repeated when new results are needed; at worst, it introduces errors. The elimination of these errors that arise due to ad hoc synchronization such as cutting and pasting is, by itself, a good recommendation for the compendium concept. And indeed, using any dynamic document system is to be encouraged, independent of the compendium concept.

It is also worth considering the role of the processing language, or transformer, that operates on the dynamic document to produce the desired output. Its primary role is to identify the relevant chunks (text or code) and to marshal these code

COBRA
A BEPRESS REPOSITORY
Collection of Biostatistics
Research Archive

http://biostats.bepress.com/bioconductor/paper2

chunks to the other relevant software components that process them appropriately. The languages that can be used in the code chunks of the dynamic documents are limited by the capabilities of the transformer. For example, the transformer can use a shell process to evaluate system-level commands, invoke Perl to execute a chunk as a Perl script, or pass the code to an R interpreter. Different transformers will support different languages for code chunks and dynamic documents can reasonably use several languages to implement the overall task.

It will be important to develop appropriate evaluation models for compendiums. For example, in the R language prototype the evaluation model is that each chunk is evaluated sequentially. All values and intermediate results from one chunk are available to any other code chunk that appears after it in the dynamic document. In this prototype there is no linking of computations between dynamic documents. For compendiums in Perl or Python, suitable evaluation models that reflect the views of those communities will need to be developed. In compendiums with mixed language code chunks, a different model may be needed.

Quality Control Software In addition to the sets of conventions and tools needed to create compendiums and to author dynamic documents, a variety of other software tools will be needed. These deal primarily with issues such as unit testing, applying version numbers and distribution.

An author needs a mechanism to verify that the code chunks perform as intended. There are many situations in which the reader will also need such validation. For example, she may want to verify that a specific static document was an output of a particular compendium. While the testing process is generally open-ended and context-specific, there are some relatively simple and achievable benchmarks. For example, we can compare output from components of the compendium with a master copy. Diagnostic checks for intermediate results (i.e. evaluating sub-groups of code chunks) are also relevant in many situations. While we are not aware of any general, widely-adopted strategies for doing this, most good software has some self-verification mechanism that can be run at installation.

Distribution Software Authors will generally want to provide readers with access to their compendiums and hence require mechanisms for distribution. Readers will need tools to help search for and locate interesting compendiums, and then to download and process the compendium for reading or other purposes. While no

COBRA
A BEPRESS REPOSITORY
Collection of Biostatistics
Research Archive

Hosted by The Berkeley Electronic Press

major system yet handles compendiums, many languages support transparent distribution and installation of modules. For example, CRAN for R and CPAN for Perl are software archives that provide the search and distribution facilities, albeit in a single centralized location. Tools provided with these languages (e.g `distutils` for Python and `install.packages` in R) provide the client-side installation mechanism that might be extended to support compendiums.

A system for attaching version numbers to compendiums will be very useful. Such a system will allow users, for example, to identify newer versions (perhaps with new data or with errors fixed), differentiate between different versions, and so on. Version numbers can be used in general distribution systems that allow users to automatically obtain updates, as the author makes them available.

5 Implementation Details

We now look at several commonly used programming languages and explore how we can make use of them to implement compendiums. We first consider and explicitly identify different roles for programming languages with respect to authoring, transforming and distributing compendiums. We focus on notions of what might be called single language compendiums, namely one that involves code chunks written for use in one language. The prototype we have discussed is a single language compendium for R and while such an approach is limiting it, can be very helpful as well. Users familiar with R and LaTeX have had little trouble using the system in a manner consistent with the general strategies and concepts we are proposing. We believe that similar gains can easily be made within other programming language communities, such as the Perl and Python communities.

A compendium is simply a software module or archive and can be organized according to accommodate processing by some specific language. We will call this language the *definition language* or perhaps the definition system. Next the dynamic documents themselves must be marked up in some language such as LaTeX or XML. We will call this the *document markup*. When a dynamic document is transformed some software language will need to be used to carry out the transformations. This could be any high level language such as R, Perl, Python or XSL; we refer to this as the *transformation language*. The actual software that carries

COBRA
A BEPRESS REPOSITORY
Collection of Biostatistics
Research Archive

out the transformations is called the *transformer* and the capabilities of the processing system are embodied here. The more general the transformer the more general the dynamic documents and hence the compendium. And finally, each of the code chunks within a dynamic document can be written in any supported language. Whether or not a dynamic document can support code chunks written in a particular language will depend on the transformer that is available.

The simplest setting is when the definition language and the transformation language are the same and the code chunks are also written in that same language. This is the case with our prototype system. Only the R language is involved. The markup language for the dynamic documents in that system is `noweb` with LaTeX for the text chunks. One could of course use a different transformer, if one were available. That transformer would need to understand the organization of the R compendiums and would need to understand the markup used for the dynamic documents, but it does not need to be written in R.

As we have mentioned previously there are many different candidate languages for carrying out each of these different roles. It is also important to emphasize that most languages can easily perform more than one role. For example, XML can be used as the markup language for the dynamic documents and it can be used to provide an organization and structure for compendiums themselves.

Perl, Python and R are widely used languages each of which has a mechanism for users to provide add-on functionality to the base system. Each provides tools for collecting the code and documentation into a module so that it can be distributed and installed on the user's machine. Additionally, each allows users to include additional files in that distribution. Table 1 identifies existing tools in these systems that can be used to carry out Steps 1 through 5 listed on page 17. This structure and extensibility provide a very natural basis for constructing compendiums. Further, the software engineering tools provided by these languages can form the basis for validity checking and other quality control procedures that the authors of compendiums will need. The path to complete implementations, while not entirely trivial, is clear. More importantly, we can see that the compendium concept fits well with different languages and is in no way specific to R.

Perl has its own language for writing documentation named POD (Plain Old Documentation) and that might be more natural for authoring Perl-based dynamic

COBRA
A BEPRESS REPOSITORY
Collection of Biostatistics
Research Archive

Hosted by The Berkeley Electronic Press

	R	Perl	Python
Document Format	XML or Sweave	XML or POD	XML
Skeleton	`package.skeleton`	`h2xs`	
Distribution unit	R package	Perl module	Python module
Distribution Mechanism	Repository tools	CPAN	Vaults of Parnassus
Installation	`R CMD INSTALL`	`perl Makefile.PL make install`	`python setup.py install`
Test Command	`R CMD check`	`make test`	`python unittest.py file`
Test Tools	`tools` package	`Test` module	`PyUnit`

Table 1: The tools in the different languages R, Perl, Python available to author, create, manage and distribute a compendium.

COBRA
A BEPRESS REPOSITORY
Collection of Biostatistics
Research Archive

documents than XML or noweb. Python's documentation mechanism is more narrowly focused and not appropriate for writing documents. As a result, it is likely that we would adopt XML or a similar language for writing Python dynamic documents.

Both Perl and Python have XML parsing capabilities which can be used to process dynamic documents authored in XML into the different chunks. Perl also provides classes for processing POD input which can be used to extract the different chunks. Regardless of how the chunks are obtained, it is straightforward to iterate over them, evaluate the code chunks, and interleave the output into the target document with the text chunks.

Both Perl and Python provide a well-defined structure for creating modules for the auxiliary software and data present in a compendium. Perl's `h2xs` tool creates a template for a module and the necessary code for installing that module. Python's `distutils` module provides facilities for specifying the contents, configuration details, etc. for a module. The resulting modules in either language can then be easily installed by the author and reader alike using the standard and simple commands `perl Makefile.PL ; make` for Perl and `python setup.py install` for Python.

Perl and Python also have well-developed modules to perform testing. In Perl these are contained in the `Test` modules, while in Python they are contained within the `PyUnit` module. Both mechanisms support test files that can be run on the installer's machine to verify that the module is working correctly. We believe that these mechanisms can be readily extended to process dynamic documents and provide verification that compendium produces the expected output.

5.1 General Comments

The tools and paradigms described in Table 1 provide support for authoring, managing and transforming the compendiums in each of the three different languages. They do not however restrict the choice of languages that can be used in the code chunks. A compendium authored in the R system could have code chunks in Perl or Python or any other language provided that the *transformer* is capable of identifying and managing that process. We note that Sweave currently only supports chunks written in R but that is a limitation due only to implementation and could

COBRA
A BEPRESS REPOSITORY
Collection of Biostatistics
Research Archive

Hosted by The Berkeley Electronic Press

be eliminated.

As indicated above there is no need for the language used to transform the dynamic documents to be the same language that was used to author the compendium. Instead a general transformation tool (e.g. XSL) which identifies the chunks itself and passes them to the appropriate processing system, e.g. R for S language code chunks and XSL rules for text chunks, can be used. Our experience with combining R and XSL (see `http://www.omegahat.org/Sxslt`) indicates that using a general transformation mechanism will often be better since it allows measure of language independence that other strategies do not. That is, the transformation tools are written once and can be applied to compendiums based on R, Perl, Python or a mixture of these (or other languages).

We also want to be careful not to gloss over the difficulties that might be involved in a general scheme. If a compendium is represented as a large XML file and is transformed by XSL, much of the work that in our prototype is carried out by R must be reimplemented. The transformer will need to do a reasonable amount of work to locate the appropriate language and to marshal the auxiliary software and data for processing. However, one gains flexibility and language independence.

6 Future Directions

In this section, we provide some discussion of what we think are likely short-term and long-term directions that are worth pursuing.

Multi-language compendiums It seems quite natural to expect that many tasks described in dynamic documents will be carried out using different pieces of software and commands from different languages. For example, one might use Perl code to collect and filter data, some system commands to combine files, and S code for statistical analysis all as parts of a single overall task. To faithfully reproduce these computations and transform the document, as the processor encounters the different code chunks, it must be able to both identify the associated software, and evaluate the code appropriately using the current inputs to obtain the output. This basically requires a markup to identify code chunks with particular processing software. The evaluation of these chunks is more problematic and requires the document processing

COBRA
A BEPRESS REPOSITORY
Collection of Biostatistics
Research Archive

system to be able to pass those chunks to other systems for evaluation. An inter-system interface is a mechanism by which one system can access functionality and evaluate code in another. Omegahat's inter-system interfaces (e.g. SJava, RSPython, RSPerl) provide these in R, while the Inline module provides similar capabilities for Perl.

Conditional Chunks Producing target documents in different formats is an important facility for the compendium. It is equally important to be able to create documents with different content for different audiences from the same compendium. For example, the output documents from a clinical trial represented as a compendium would naturally include interim reports, final reports and specialized reports for safety and data monitoring, each of which are created at different times during the evolution of the trial and intended for different readers. Such multi-targeted output requires support both in the authoring and the processing of the dynamic documents to handle the text chunks in a richer and more complex manner, e.g. conditional inclusion in the target document. Again markup for specifying attributes for text chunks is needed.

Some dynamic documents may provide alternative implementations for several languages. For example, an author might offer code for R and Matlab to appeal to a wider audience. In such cases, the processing system should be parameterizable to identify the target language of interest and ignore the code for the alternative system(s).

Interactivity The advent of the Web and browsers has familiarized many of us with interactive content within documents, e.g. HTML forms for data input and sliders for controlling a display. One of the uses of compendiums is pedagogy and providing interactive facilities with which the readers of the output documents can manipulate and control the computations directly will be a powerful facility. This is yet another type of transformation of the dynamic document with a different intended readership. Sawitzki (2002) describes a system for creating interactive documents and discusses some of the issues involved.

COBRA
A BEPRESS REPOSITORY
Collection of Biostatistics
Research Archive

Hosted by The Berkeley Electronic Press

Metadata Inclusion of programmatically accessible meta-information in documents facilitates both richer interactions and better descriptions of the content. Many scientific documents contain keywords as part of the text. Making these explicitly available to cataloging and indexing software as programmatically extractable elements of the dynamic document will facilitate richer distribution services. Since dynamic documents are software, licensing also becomes pertinent. One may wish to restrict evaluation or access to data within the compendium. This can be done with meta-information such as license key matching or explicit code within the document to verify authorization. Another use of meta-information is the inclusion of digital signatures which can be used to verify the origin and legitimacy of the compendium.

6.1 XML

Literate programming is a major component in the concept of compendiums. Dynamic documents as described in earlier sections are a relatively straightforward extension of literate programming. The extensions outlined in the previous paragraphs introduce both technical and conceptual challenges to the traditional literate programming paradigm. Adding attributes to text chunks (e.g. target languages, conditional evaluation,) changes the underlying model of `noweb` and `Sweave`. The addition of meta-information moves away from the two type chunk format (text and code) to allowing many different types of document elements. While we could adapt `noweb` and `Sweave` to handle each of these extensions, we feel that adopting a widely-used and extensible markup language will be a more advantageous strategy. The natural candidate is XML.

Over the past several years, XML has become the standard for document and data representation and interchange within and between a multitude of different and diverse communities. Not only is XML a *de facto* standard, it has a well-documented, formal specification. Since XML is a general, extensible markup language framework, it is sufficient for specifying all the elements of any dynamic document. It is also extensible and so allows us and others to introduce new tags or chunk types and attributes for the different pieces of the document, as needed.

The fact that XML is in widespread use has two immediate consequences.

COBRA
A BEPRESS REPOSITORY
Collection of Biostatistics
Research Archive

http://biostats.bepress.com/bioconductor/paper2

There is a large, knowledgeable user-base and an active development community. Through the work of this development community, there is a vast collection of both low- and high-level tools for working with XML. These include software for authoring, transforming and validating XML documents, available in almost every programming language. Hence, we neither need to specify or invent the general markup structure, nor develop all required tools to process dynamic documents as much of this work has already been done or is being done by others. These same advantages are available to developers of compendiums in other languages such as Perl and Python. This common structure provides a unification across the languages and also the various communities which is essential for disseminating statistical ideas. In addition to the infra-structural tools, there are several existing higher-level frameworks for processing technical documents such as DocBook and Jade. More related to the statistical community, we have provided general support for XML in S that makes it possible to both read and write XML from the S system. Also, S can be combined with XSL by either embedding the XSL transformer in R or extending the XSL engine to be able to call S functions (Temple Lang (2001) and `www.omegahat.org/Sxslt`).

Data play an important role in our concept of the compendium. Our previous considerations suggested including the datasets as files within the compendium, i.e. within the specific packaging mechanism. In some cases, it is simpler to be able to include the data directly within a dynamic document. General processing tools will only be able to interpret these data if they are self-describing according to some standard which does not rely on ad hoc conventions. The data representation aspects of XML again make this easy. For example, we might include a dataset encoded using SOAP, an image using SVG or a graph using GXL. This makes the same data available to all languages that are being used.

6.2 Practical Issues

One can argue that compendiums would be good accompaniments to papers submitted to journals or that compendiums should be the preferred form of submission. Currently, referees of papers that are largely based on computation can face substantial difficulties. They are often faced with the dilemma of either doing a great deal of work to reproduce the authors computations or they must take on faith what

COBRA
A BEPRESS REPOSITORY
Collection of Biostatistics
Research Archive

Hosted by The Berkeley Electronic Press

the authors report. When these referees are provided with compendiums, their job will be much easier. They can explore the dynamic documents, see which computations were carried out, and answer the specific questions of when and how. Compendiums offer great opportunities and should substantially ease the burden of refereeing such papers in detail.

A secondary issue that should also be addressed is to understand the role or purpose of compendiums. Are they software implementations that should work on a variety of inputs? The answer to this is an emphatic no. That is not the intention of such a device. The purpose of the compendium concept is to provide support (by reproducing the output) for the claims made in a scientific article. The production of high quality software is a related but very different task.

Another practical issue that will arise is the problem of proprietary content. It is reasonably common for authors to be able to disseminate results of an analysis or descriptions of an algorithm but not to be able to make the data or the software available. This is already a problem for peer-reviewed publication. Undoubtedly, data and code are vital components of an actual executable compendium. However, the absence of either from the compendium still leaves available many more of the details for the referee or reader than a regular static document. However, even in cases where only the static document is submitted for publication, there are benefits to the author from creating and working with a compendium. If questions arise in the future or if new data are made available their inclusion is much easier if a compendium has been created.

The richer concept of an executable document that we have described is likely to support a form of limited or restricted execution. Digital signatures or "license managers" that allow only privileged recipients to re-run the computations appear to be a practical extension.

It is also worth noting that publication is quite a broad term. And within that compendiums have general applicability. An author might publish their work by sending it to handful of close associates. One post-doc in a lab might publish their work by making a compendium available to their lab-mates. In all cases the purpose is the same – to enable the reader to comprehend and reproduce the results of the author. In all cases it is a valuable tool, there is no need for publication in journals or widespread distribution for the compendium concept to be worthwhile.

COBRA
A BEPRESS REPOSITORY
Collection of Biostatistics
Research Archive

http://biostats.bepress.com/bioconductor/paper2

As another example, cancer clinical trials often extend for many years and the study statistician often changes several times before the research is complete. In many cases, one may want to revisit an analysis at a later date. In this setting, a compendium will make it easier for other statisticians to take on the study and for the results to be used in future studies. But again there is no need or reason to publish the whole compendium; portions of it may appear in interim reports or in final publications.

The lifetime of a compendium may be limited. As programming languages evolve or disappear it may not be possible to ensure that all existing compendiums are runnable. However, the same is true for any electronic document format; will Word or PDF exist forever? The answer, of course, is no. But, it is often the case that software tools that aid in migration are developed and used for formats that are popular. We would also like to point out that even so, we will be no worse off than we are now with static papers. And in the event that a reconstruction is necessary, it will be much easier to do so from a compendium than from a paper.

We would also like to indicate that we are cognizant of the security issues that will be involved in obtaining and running compendiums. In essence we are proposing that users download and run programs on their local machine. There is some potential for malicious behavior and research into methods that may help to make the process more secure is certainly warranted. This is currently no more of an issue than using code from an R package or Perl module downloaded from the Internet. However, it does illustrate the need to provide some *sandbox* mechanism in which the compendium can be processed with secure limitations.

7 Discussion

As noted, our focus in this paper has been on the concept of the compendium and to point out that sufficient tools exist to build and distribute them now. We have intentionally not gone into details about the process of authoring such a document or different aspects of rendering it for the reader. We plan to explore these issues in our future research.

In this multi-media era, it seems appropriate to consider interactive documents which allow the reader to control different aspects of the computations (e.g. adjust-

COBRA
A BEPRESS REPOSITORY
Collection of Biostatistics
Research Archive

Hosted by The Berkeley Electronic Press

ing a tuning parameter via a slider widget) or even introduce their own data. Such interactivity has been discussed in Temple Lang (2001) and Sawitzki (2002). These concepts are also being explored in the StatDocs project (`www.statdocs.org`). Much more development will be needed as we learn what does and does not work and what is and is not useful. These explorations will lead to new tools and improvements to old ones. However, we feel that currently available technology provides a structure that can support a much richer environment than is currently available.

The ideas put forward in this article are mainly aggregations based on other tools and we consider the implications when these methods are applied to the concept of reproducible research. What is remarkable is that using very few tools and conventions, we have been able to construct a workable system. Granted that system is too reliant on a single engine (R) to be generally viable, but it seems to be a more than adequate starting point.

The compendium concept, and that of reproducible research, has the potential to improve the state of publication about computational science. The tools we have proposed and discussed will allow us to move from an era of advertisement to one where our scholarship itself is published. This exposes the computations themselves to the scientific method and enhances the potential for iterative refinement and extension.

Acknowledgments

We would like to thank the R Core Development Team for providing an intellectual environment that stimulates research. We particularly thank Vincent Carey, Friedrich Leisch and Kurt Hornik for helpful discussions and advice. We would also like to thank an Associate Editor and two referees for their patience and insightful comments.

Glossary

The following provides a description and some references for some of the software and standards referenced in the paper.

COBRA
A BEPRESS REPOSITORY
Collection of Biostatistics
Research Archive

AbiWord is the Gnome word processor, (http://www.abiword.org)

Emacs is the GNU text editor (and more), www.emacs.org.

GXL The Graph eXchange Language (GXL) is an XML-based representation for describing graphs, i.e. nodes and connecting edges. It is used in a variety of different domains ranging from Bioinformatics to software engineering. See http://www.gupro.de/GXL/ for more information.

Perl (Practical Extraction and Report Language) is a general high-level programming language that excels at text manipulation. See http://www.perl.org and the Comprehensive Perl Archive Network (CPAN) for available add-ons to the system.

Python is another high-level scripting language, more structured than Perl with an increasing user-base and collection of contributed extension modules. More information is available from http://www.python.org.

R is an Open Source implementation of the ACM award winning S language and similar to the commercial implementation S-Plus. S is both a general programming language and an extensible interactive environment for data analysis and graphics. See http://www.r-project.org for information on the project and CRAN (the Comprehensive R Archive Network) http://cran.r-project.org for available software and packages.

SOAP is an acronym for the Simple Object Access Protocol which is an XML dialect for representing distributed or remote method calls between applications. It has become a very popular protocol for implementing Web services, using HTTP as the communication mechanism and XML as the data representation. See http://www.w3.org/TR/SOAP/ for more information.

SVG Scalable Vector Graphics is another XML dialect used to represent two-dimensional graphical displays. It provides a way to describe drawing operations for graphics objects as well as interactivity in the form of event action descriptions and animations. More details are available from http://www.w3.org/Graphics/SVG.

COBRA
A BEPRESS REPOSITORY
Collection of Biostatistics
Research Archive

Hosted by The Berkeley Electronic Press

XML stands for the eXtensible Markup Language, a text-based markup mechanism for representing self-describing data. Its syntax is the same as the familiar HTML (the Hyper Text Markup Language). However, one can define new and arbitrary tags or elements in XML to define new and different specialized dialects for representing arbitrary data in a self-describing manner. XML documents are made up of nodes which are arranged hierarchically. A class of XML documents (i.e. a dialect) can be described symbolically via a Document Type Definition (DTD) which describes the possible relationships between different types of nodes, i.e. which nodes can be nested within other node types and in what order. This allows one to also validate XML documents according to this specification without actually interpreting the specific content. Schema are a newer way to provide information not just about the structure of the document, but also about the data types within XML nodes.

The W3 organization (`http://www.w3.org`) provides much of the standardization and specification of XML and its dialects. The Cover Pages Web site (`http://www.coverpages.org`) provides information on using XML in a wide variety of different applications.

XSL (the eXtensible Stylesheet Language) is a specific XML dialect that is used to describe transformations that map an XML document to an other XML document or different format. Typically, an XSL transformer (XSLT) is used to apply a stylesheet to an XML document.

References

Baggerly, K. A., Morris, J. S., and Coombes, K. R. (2004). Reproducibility of seldi-tof protein patterns in serum: comparing datasets from different experiments. *Bioinformatics*, 20:777–85.

Buckheit, J. and Donoho, D. L. (1995). Wavelab and reproducible research. In Antoniadis, A., editor, *Wavelets and Statistics*. Springer-Verlag.

COBRA
A BEPRESS REPOSITORY
Collection of Biostatistics
Research Archive

http://biostats.bepress.com/bioconductor/paper2

Carey, V. J. (2001). Literate statistical programming: concepts and tools. *Chance*, 14:46–50.

Gentleman, R. (2004). Reproducible research: A bioinformatics case study. *Statistical Applications in Genetics and Molecular Biology*, page to appear.

Ihaka, R. and Gentleman, R. (1996). R: A language for data analysis and graphics. *Journal of Computational and Graphical Statistics*, 5:299–314.

Knuth, D. (1992). *Literate Programming*. Center for the Study of Language and Information, Stanford, California.

Leisch, F. (2002). Sweave: Dynamic generation of statistical reports using literate data analysis. In Härdle, W. and Rönz, B., editors, *Compstat 2002 — Proceedings in Computational Statistics*, pages 575–580. Physika Verlag, Heidelberg, Germany. ISBN 3-7908-1517-9.

Oram, A. and Talbott, S. (1991). *Managing Projects with make*. O'Reilly.

R Development Core Team (1999). *Writing R Extensions*. The R Foundation, Vienna, Austria, 1.8 edition.

Ramsey, N. (1994). Literate programming simplified. *IEEE Software*, 11(5):97–105.

Rossini, A. (2001). Literate statistical analysis. In Hornik, K. and Leisch, F., editors, *Proceedings of the 2nd International Workshop on Distributed Statistical Computing, March 15-17, 2002.* http://www.ci.tuwien.ac.at/Conferences/DSC-2001/Proceedings.

Rossini, A. J., Heiberger, R. M., Sparapani, R. A., Maechler, M., and Horniki, K. (2004). Emacs speaks statistics: A multiplatform, multipackage development environment for statistical analysis. *Journal of Computational and Graphical Statistics*, 13:247–261.

Sawitzki, G. (2002). Keeping statistics alive in documents. *Computational Statistics*, 17:65–88.

COBRA
A BEPRESS REPOSITORY
Collection of Biostatistics
Research Archive

Hosted by The Berkeley Electronic Press

Temple Lang, D. (2001). Embedding S in other languages and environments. In Hornik, K. and Leisch, F., editors, *Proceedings of the 2nd International Workshop on Distributed Statistical Computing, March 15-17, 2002.* http://www.ci.tuwien.ac.at/Conferences/DSC-2001/Proceedings.

http://biostats.bepress.com/bioconductor/paper2