Rchive: Towards Provenance Tracking in R

Andrew Runnalls

Computing Laboratory, University of Kent, UK

What is Provenance?

From the Oxford English Dictionary:

provenance, *n.*

1. The proceeds from a business. *Obs. rare.*
2. The fact of coming from some particular source or quarter; origin, derivation.
3. The history of the ownership of a work of art or an antique, used as a guide to authenticity or quality; a documented record of this.
4. *Forestry.* The geographic source of tree seed; the place of origin of a tree. Also: seed from a specific location.

What is Provenance?

From the Oxford English Dictionary:

provenance, *n.*

1. The proceeds from a business. *Obs. rare.*
2. The fact of coming from some particular source or quarter; origin, derivation.
3. The history of the ownership of a work of art or an antique, used as a guide to authenticity or quality; a documented record of this.
4. *Forestry.* The geographic source of tree seed; the place of origin of a tree. Also: seed from a specific location.

In information systems, ‘provenance’ is used following the third sense above: In producing a given data object, what primary sources of data were drawn upon, and what sequence of operations was applied to those primary data sources?

Provenance-Aware Computing

There is increasing interest in keeping track of the provenance of data within information systems. The second International Provenance and Annotation Workshop (IPAW 2008), held at Salt Lake City in June this year, attracted 55 delegates, and the presented papers included applications to the following fields:

- proteomics;
- disease diagnosis using information retrieval systems;
- handling of data from biological imaging (e.g. MRI, fMRI, CT, DTI, PET);
- pulmonary immunity modelling;
- gene sequencing and phylogenetic analysis;
- computer systems, including network routing, grid computing, and intrusion detection;
- earth sciences, particularly handling remote sensing data and other images;
- remote health monitoring;
- hydrology and oceanography;
- weather monitoring;
- atmospheric chemistry.

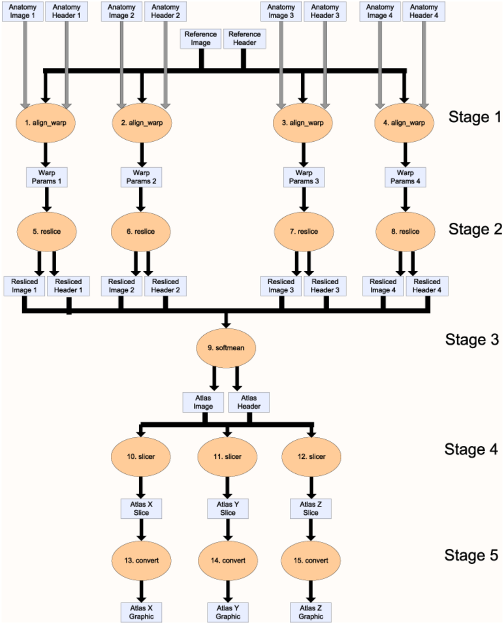
The First Provenance Challenge

The First Provenance Challenge[1] emerged from IPAW2006 (Chicago), and was intended to explore and compare the provenance-tracking capabilities of various systems for managing scientific workflows. (Examples of such systems are ES3, Kepler, Taverna and VisTrails.)

The challenge posed a problem of aggregating images obtained by function Magnetic Resonance Imaging (fMRI) into a population-averaged ‘brain atlas’.

[1] http://twiki.ipaw.info/bin/view/Challenge/FirstProvenanceChallenge

The First Provenance Challenge



1. Compare a new 3D brain image to a reference image to decide how the new image should be ‘warped’ (spatially transformed) to match the reference. (`align_warp`)
2. Carry out the image warping. (`reslice`)
3. Average all the images into a single 3D image. (`softmean`)
4. Extract 2D slices from the 3D image perpendicular to each of the *x*, *y* and *z* axes. (`slicer`)
5. Convert the 2D slices into a standard computer-graphics image format. (`convert`)

The First Provenance Challenge

Example queries

- Find the process that led to a particular output 2D image, i.e. everything that caused image to be as it is. This should tell us the new brain images from which the averaged atlas was generated, the warping performed etc.
- Find all invocations of the warping procedure that used a particular model (a twelfth-order nonlinear model with 1365 parameters) that ran on a Monday.
- A user has annotated some input images with a key-value pair `center=UChicago`. Find the outputs of `align_warp` where the inputs are annotated with `center=UChicago`.
- A user has annotated some atlas graphics with a key-value pair where the key is `studyModality`. Find all the graphical atlas sets that have metadata annotation studyModality with values speech, visual or audio, and return all other annotations to these files.

The First Provenance Challenge

Example queries

- Find the process that led to a particular output 2D image, i.e. everything that caused image to be as it is. This should tell us the new brain images from which the averaged atlas was generated, the warping performed etc.
- Find all invocations of the warping procedure that used a particular model (a twelfth-order nonlinear model with 1365 parameters) that ran on a Monday.
- A user has annotated some input images with a key-value pair `center=UChicago`. Find the outputs of `align_warp` where the inputs are annotated with `center=UChicago`.
- A user has annotated some atlas graphics with a key-value pair where the key is `studyModality`. Find all the graphical atlas sets that have metadata annotation studyModality with values speech, visual or audio, and return all other annotations to these files.

The First Provenance Challenge

Example queries

- Find the process that led to a particular output 2D image, i.e. everything that caused image to be as it is. This should tell us the new brain images from which the averaged atlas was generated, the warping performed etc.
- Find all invocations of the warping procedure that used a particular model (a twelfth-order nonlinear model with 1365 parameters) that ran on a Monday.
- A user has annotated some input images with a key-value pair `center=UChicago`. Find the outputs of `align_warp` where the inputs are annotated with `center=UChicago`.
- A user has annotated some atlas graphics with a key-value pair where the key is `studyModality`. Find all the graphical atlas sets that have metadata annotation studyModality with values speech, visual or audio, and return all other annotations to these files.

The Second Provenance Challenge and the OPM

The Second Provenance Challenge took place over the period November 2006 to June 2007, and explored the extent to which the provenance information generated by various systems was intertranslatable.

A workshop at Salt Lake City in August 2007 discussed the results of the second challenge, and led to the drafting of the **Open Provenance Model** (OPM), as a means “to allow provenance information to be exchanged between systems, by means of a compatibility layer based on a shared provenance model”. Version 1.01 of the OPM was released in July 2008.

A third challenge is currently being formulated, to explore the strengths and weaknesses of the OPM.

The Second Provenance Challenge and the OPM

The Second Provenance Challenge took place over the period November 2006 to June 2007, and explored the extent to which the provenance information generated by various systems was intertranslatable.

A workshop at Salt Lake City in August 2007 discussed the results of the second challenge, and led to the drafting of the **Open Provenance Model** (OPM), as a means "to allow provenance information to be exchanged between systems, by means of a compatibility layer based on a shared provenance model". Version 1.01 of the OPM was released in July 2008.

A third challenge is currently being formulated, to explore the strengths and weaknesses of the OPM.

The Second Provenance Challenge and the OPM

The Second Provenance Challenge took place over the period November 2006 to June 2007, and explored the extent to which the provenance information generated by various systems was intertranslatable.

A workshop at Salt Lake City in August 2007 discussed the results of the second challenge, and led to the drafting of the **Open Provenance Model** (OPM), as a means “to allow provenance information to be exchanged between systems, by means of a compatibility layer based on a shared provenance model”. Version 1.01 of the OPM was released in July 2008.

A third challenge is currently being formulated, to explore the strengths and weaknesses of the OPM.

S AUDIT

One of the pioneer provenance-aware applications was S, with its AUDIT facility: the classic paper *Auditing of Data Analyses*[2] by Becker and Chambers is widely cited in the P-A literature.

An S session would maintain an **audit file**, recording all the top-level commands issued in this and previous sessions within the workspace, and identifying the data objects read and modified by the commands.

[2] SIAM J. Sci. Stat. Comput. 9 [1988] pp. 747–60

An S Audit File

```
#~New session: Time: 542034997; Version: "S Tue Mar 3 10:14:20 EST 1987"
m<-matrix(read("brain.body"),byrow=T,ncol=2)
#~put "/usr/rab/.Data/m" 542035057 "structure"
brain<-m[,1]
#~get "/usr/rab/.Data/m" 542035057 "any"
#~put "/usr/rab/.Data/brain" 542035066 "real"
body<-m[,2]
#~get "/usr/rab/.Data/m" 542035057 "any"
#~put "/usr/rab/.Data/body" 542035072 "real"
plot(body,brain)
#~get "/usr/rab/.Data/body" 542035072 "any"
#~get "/usr/rab/.Data/brain" 542035066 "any"
```

(from Becker and Chambers [1988] p. 754, slightly edited)

An S Audit File

```
#~New session: Time: 542034997; Version: "S Tue Mar 3 10:14:20 EST 1987"
m<-matrix(read("brain.body"),byrow=T,ncol=2)
#~put "/usr/rab/.Data/m" 542035057 "structure"
brain<-m[,1]
#~get "/usr/rab/.Data/m" 542035057 "any"
#~put "/usr/rab/.Data/brain" 542035066 "real"
body<-m[,2]
#~get "/usr/rab/.Data/m" 542035057 "any"
#~put "/usr/rab/.Data/body" 542035072 "real"
plot(body,brain)
#~get "/usr/rab/.Data/body" 542035072 "any"
#~get "/usr/rab/.Data/brain" 542035066 "any"
```

Recorded S command

(from Becker and Chambers [1988] p. 754, slightly edited)

An S Audit File

```
#~New session: Time: 542034997; Version: "S Tue Mar 3 10:14:20 EST 1987"
m<-matrix(read("brain.body"),byrow=T,ncol=2)
#~put "/usr/rab/.Data/m" 542035057 "structure"
brain<-m[,1]
#~get "/usr/rab/.Data/m" 542035057 "any"
#~put "/usr/rab/.Data/brain" 542035066 "real"
body<-m[,2]
#~put "/usr/rab/.Data/body" 542035072 "real"
plot(body,brain)
#~get "/usr/rab/.Data/body" 542035072 "any"
#~get "/usr/rab/.Data/brain" 542035066 "any"
```

Object read by command

Time of object creation

(from Becker and Chambers [1988] p. 754, slightly edited)

An S Audit File

```
#~New session: Time: 542034997; Version: "S Tue Mar 3 10:14:20 EST 1987"
m<-matrix(read("brain.body"),byrow=T,ncol=2)
#~put "/usr/rab/.Data/m" 542035057 "structure"
brain<-m[,1]
#~get "/usr/rab/.Data/m" 542035057 "any"
#~put "/usr/rab/.Data/brain" 542035066 "real"
body<-m[,2]
#~get "/usr/rab/.Data/m" 542035057 "any"
#~put "/usr/rab/.Data/body" 542035072 "real"
plot(body,brain)
#~get "/usr/rab/.Data/body" 542035072 "any"
#~get "/usr/rab/.Data/brain" 542035066 "any"
```

Object created by command

Time of object creation

(from Becker and Chambers [1988] p. 754, slightly edited)

An S Audit File

```
#~New session: Time: 542034997; Version: "S Tue Mar 3 10:14:20 EST 1987"
m<-matrix(read("brain.body"),byrow=T,ncol=2)
#~put "/usr/rab/.Data/m" 542035057 "structure"
brain<-m[,1]
#~get "/usr/rab/.Data/m" 542035057 "any"
#~put "/usr/rab/.Data/brain" 542035066 "real"
body<-m[,2]
#~get "/usr/rab/.Data/m" 542035057 "any"
#~put "/usr/rab/.Data/body" 542035072 "real"
plot(body,brain)
#~get "/usr/rab/.Data/body" 542035072 "any"
#~get "/usr/rab/.Data/brain" 542035066 "any"
```

(from Becker and Chambers [1988] p. 754, slightly edited)

S was provided with a tool `S AUDIT` which examined an audit file, and could be used to interrogate the provenance of particular data objects, show which statements used particular objects, or generate a script to recreate a particular object.

The CXXR Project

The aim of the CXXR project[3] is progressively to reengineer the fundamental parts of the R interpreter from C into C++, with the intention that:

- Full functionality of the standard R distribution is preserved;
- The behaviour of R code is unaffected (unless it probes into the interpreter internals);
- The primary interfaces between the interpreter and C and Fortran code are as far as possible unaffected.

Work started in May 2007, shadowing R-2.5.1; the current release (tested on Linux and Mac OS X) shadows R-2.7.1.

CXXR is intended to make it easier to produce experimental versions of the R interpreter.

[3] www.cs.kent.ac.uk/projects/cxxr

The CXXR Project

The aim of the CXXR project[3] is progressively to reengineer the fundamental parts of the R interpreter from C into C++, with the intention that:

- Full functionality of the standard R distribution is preserved;
- The behaviour of R code is unaffected (unless it probes into the interpreter internals);
- The primary interfaces between the interpreter and C and Fortran code are as far as possible unaffected.

Work started in May 2007, shadowing R-2.5.1; the current release (tested on Linux and Mac OS X) shadows R-2.7.1.

CXXR is intended to make it easier to produce experimental versions of the R interpreter.

[3] www.cs.kent.ac.uk/projects/cxxr

The CXXR Project

The aim of the CXXR project[3] is progressively to reengineer the fundamental parts of the R interpreter from C into C++, with the intention that:

- Full functionality of the standard R distribution is preserved;
- The behaviour of R code is unaffected (unless it probes into the interpreter internals);
- The primary interfaces between the interpreter and C and Fortran code are as far as possible unaffected.

Work started in May 2007, shadowing R-2.5.1; the current release (tested on Linux and Mac OS X) shadows R-2.7.1.

CXXR is intended to make it easier to produce experimental versions of the R interpreter.

[3] www.cs.kent.ac.uk/projects/cxxr

Rchive

The primary motivation behind CXXR is to produce a variant of the R interpreter—designated **Rchive**—which tracks the provenance of the objects it manipulates.

Rchive doesn't exist yet: we're at the requirements-gathering stage. . . and that's the purpose of this talk.

Provenance Priorities for Rchive

In decreasing order of priority:

1. *Preserve* all primary data . . . and the associated metadata.
2. Be able to *identify* all R objects derived in some way from a given object. (Remember that R objects include functions as well as data.)
3. Be able to *reproduce* a derived object from its precursors (possibly having corrected or adjusted these precursors).

Provenance Priorities for Rchive

In decreasing order of priority:

1. *Preserve* all primary data . . . and the associated metadata.
2. Be able to *identify* all R objects derived in some way from a given object. (Remember that R objects include functions as well as data.)
3. Be able to *reproduce* a derived object from its precursors (possibly having corrected or adjusted these precursors).

Provenance Priorities for Rchive

In decreasing order of priority:

1. *Preserve* all primary data . . . and the associated metadata.
2. Be able to *identify* all R objects derived in some way from a given object. (Remember that R objects include functions as well as data.)
3. Be able to *reproduce* a derived object from its precursors (possibly having corrected or adjusted these precursors).

A Changed Scene

Things have moved on since the early days of S. In making R provenance-aware, we need to take account of the following:

- The vast number of add-on packages and libraries now available for R (including packages originally designed for S/S-plus);
- Source data are increasingly derived from databases and online resources rather than local files;
- It is desirable to be interoperable with other provenance-aware applications.

A Changed Scene

Things have moved on since the early days of S. In making R provenance-aware, we need to take account of the following:

- The vast number of add-on packages and libraries now available for R (including packages originally designed for S/S-plus);
- Source data are increasingly derived from databases and online resources rather than local files;
- It is desirable to be interoperable with other provenance-aware applications.

A Changed Scene

Things have moved on since the early days of S. In making R provenance-aware, we need to take account of the following:

- The vast number of add-on packages and libraries now available for R (including packages originally designed for S/S-plus);
- Source data are increasingly derived from databases and online resources rather than local files;
- It is desirable to be interoperable with other provenance-aware applications.

The Open Provenance Model

Entities

The OPM represents provenance as a directed acyclic graph, with the following node types:

Artifact Immutable piece of state, which may have a physical embodiment in a physical object, or a digital representation in a computer system.

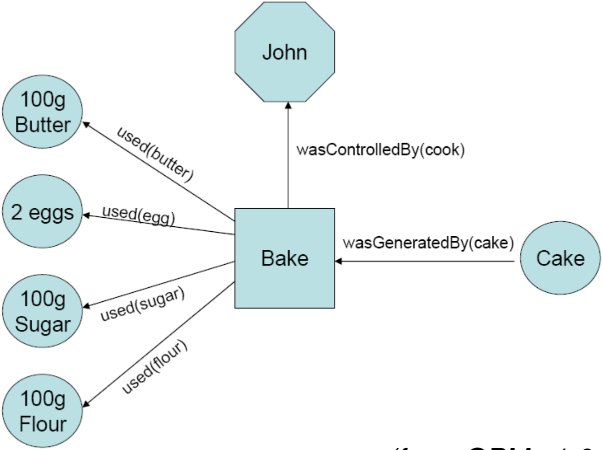
Process Action or series of actions performed on or caused by artifacts, and resulting in new artifacts.

Agent Contextual entity acting as a catalyst of a process, enabling, facilitating, controlling, affecting its execution.

(OPM v1.01)

The Open Provenance Model

Example



Note that the ‘100g butter’ node refers to *a particular 100 grams* of butter, and so on: we want, for example, to be able to trace the cakes made from a rancid pack of butter.

(from OPM v1.01)

OPM Entities in Rchive

Artifact In Rchive, all R objects (as listed by `ls()` or `objects()`) will be considered to be artifacts. *NB: This includes functions.*

Process In Rchive, a top-level R command invocation will be considered to be a process.

Agent In Rchive, each process will be associated with a user name.

OPM Entities in Rchive

Artifact
: In Rchive, all R objects (as listed by `ls()` or `objects()`) will be considered to be artifacts. *NB: This includes functions.*

Process
: In Rchive, a top-level R command invocation will be considered to be a process.

Agent
: In Rchive, each process will be associated with a user name.

OPM Entities in Rchive

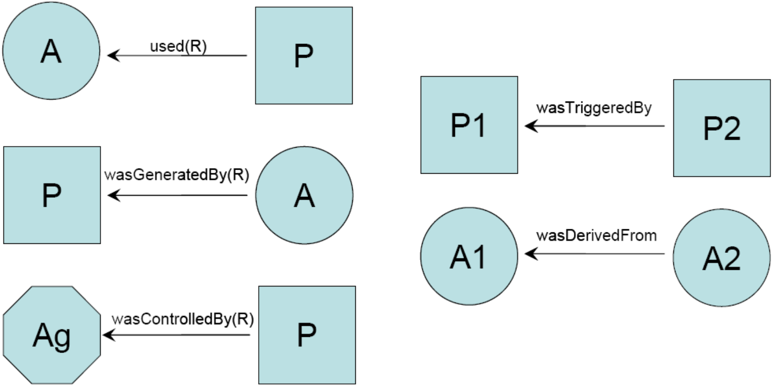
Artifact In Rchive, all R objects (as listed by `ls()` or `objects()`) will be considered to be artifacts. *NB: This includes functions.*

Process In Rchive, a top-level R command invocation will be considered to be a process.

Agent In Rchive, each process will be associated with a user name.

The Open Provenance Model

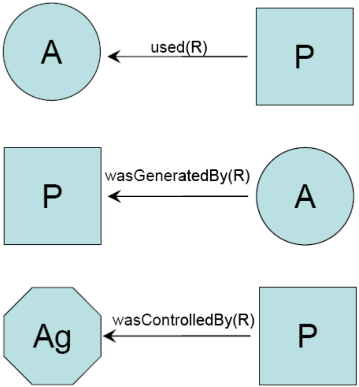
Annotated edges

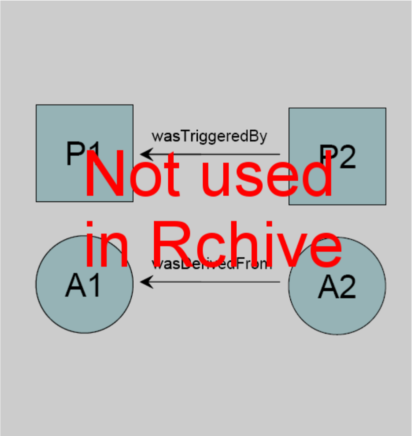


(from OPM v1.01)

The Open Provenance Model

Annotated edges



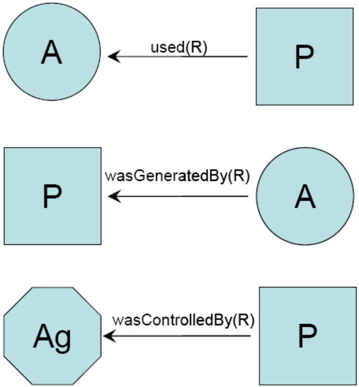


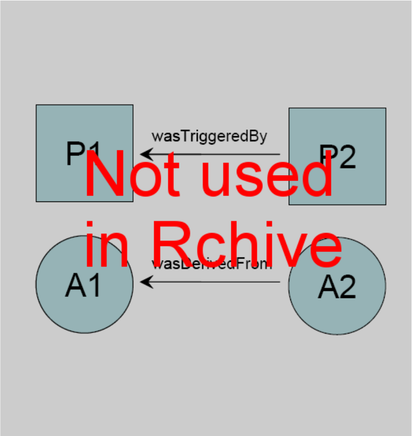
Not used in Rchive

(from OPM v1.01)

The Open Provenance Model

Annotated edges



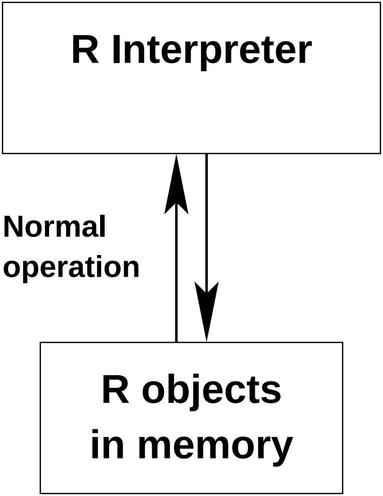


Not used in Rchive

(from OPM v1.01)

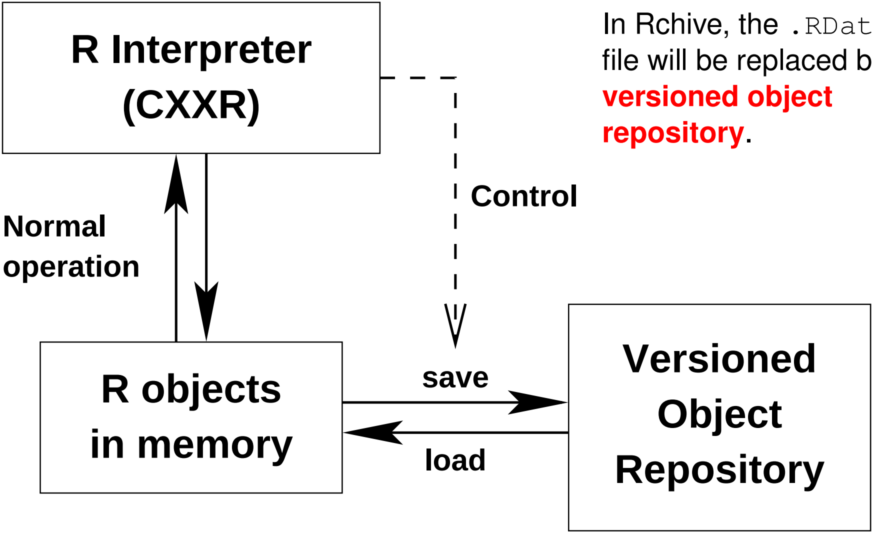
OPM recommends that the edges associated with a particular process be labelled with distinct **roles**: `R` in the edge annotations. But it's doubtful whether Rchive will be able to make meaningful use of roles.

Architecture of Rchive



Whereas S (and S-plus) store data objects in individual files, R normally stores data objects in memory: typically a workspace file (`.RData`) containing all the data objects is loaded at the start of a session and saved at the end.

Architecture of Rchive



In Rchive, the `.RData` file will be replaced by a **versioned object repository**.

Save Logic

When Rchive is requested to save an object, `x` say, it saves the following to the versioned object repository:

- `x` itself, along with a ‘universally unique identifier’ (UUID) identifying this particular version of `x`.
- An audit trail (*à la* S AUDIT) tracing the R commands used to derive `x` from versioned objects already in the repository.
- Any **tainted objects** encountered on that audit trail.

Save Logic

When Rchive is requested to save an object, `x` say, it saves the following to the versioned object repository:

- `x` itself, along with a ‘universally unique identifier’ (UUID) identifying this particular version of `x`.
- An audit trail (*à la* S AUDIT) tracing the R commands used to derive `x` from versioned objects already in the repository.
- Any **tainted objects** encountered on that audit trail.

Save Logic

When Rchive is requested to save an object, `x` say, it saves the following to the versioned object repository:

- `x` itself, along with a ‘universally unique identifier’ (UUID) identifying this particular version of `x`.
- An audit trail (*à la* S AUDIT) tracing the R commands used to derive `x` from versioned objects already in the repository.
- Any **tainted objects** encountered on that audit trail.

Tainted Objects

An object created or modified by a top-level R command is considered **tainted** if it cannot be reliably reproduced from its precursors simply by rerunning the command.

This may be because:

- The command invokes non-R code (e.g. C or FORTRAN) whose provenance cannot be accredited;
- The command involves user interaction (e.g. `edit()` or `identify()`), or accesses some external resource which does not perform provenance-tracking in an accredited way.
- The command explicitly marks its output as tainted.

(This approach reflects the handling of ‘external services’ within the NRC dataflow model of Hidders *et al.* [2007].)

Tainted Objects

An object created or modified by a top-level R command is considered **tainted** if it cannot be reliably reproduced from its precursors simply by rerunning the command.

This may be because:

- The command invokes non-R code (e.g. C or FORTRAN) whose provenance cannot be accredited;
- The command involves user interaction (e.g. `edit()` or `identify()`), or accesses some external resource which does not perform provenance-tracking in an accredited way.
- The command explicitly marks its output as tainted.

(This approach reflects the handling of ‘external services’ within the NRC dataflow model of Hidders *et al.* [2007].)

R Packages

By default, any data objects that result from executing non-R code within loaded packages will be regarded as tainted.

(Possibly this will apply even for data objects generated by R code within packages.)

A priority is to devise a mechanism whereby a package can present credentials to Rchive, in the light of which Rchive may decide to a treat the package (or some of its components) as providing a **durable resource**, i.e. a resource whose behaviour can be reliably reproduced in the future.

If a package component is considered to be a durable resource, then data objects generated using it need not be marked as tainted.

R Packages

By default, any data objects that result from executing non-R code within loaded packages will be regarded as tainted.

(Possibly this will apply even for data objects generated by R code within packages.)

A priority is to devise a mechanism whereby a package can present credentials to Rchive, in the light of which Rchive may decide to a treat the package (or some of its components) as providing a **durable resource**, i.e. a resource whose behaviour can be reliably reproduced in the future.

If a package component is considered to be a durable resource, then data objects generated using it need not be marked as tainted.

Data from External Sources

By default, any data obtained from a source outside R (e.g. a local file or an online database) will be regarded as tainted, and will be preserved as part of the audit trail of any saved object that depends on it.

However, it is not the raw data that is preserved, but only the output of the top-level R command that accesses the resource. This allows some preliminary data condensation to take place.

As with packages, it would be helpful to have a mechanism to designate an external data source (e.g. the Dataverse Network Project[4]) as a durable resource.

[4] http://thedata.org

Data from External Sources

By default, any data obtained from a source outside R (e.g. a local file or an online database) will be regarded as tainted, and will be preserved as part of the audit trail of any saved object that depends on it.

However, it is not the raw data that is preserved, but only the output of the top-level R command that accesses the resource. This allows some preliminary data condensation to take place.

As with packages, it would be helpful to have a mechanism to designate an external data source (e.g. the Dataverse Network Project[4]) as a durable resource.

[4] http://thedata.org

Data from External Sources

By default, any data obtained from a source outside R (e.g. a local file or an online database) will be regarded as tainted, and will be preserved as part of the audit trail of any saved object that depends on it.

However, it is not the raw data that is preserved, but only the output of the top-level R command that accesses the resource. This allows some preliminary data condensation to take place.

As with packages, it would be helpful to have a mechanism to designate an external data source (e.g. the Dataverse Network Project[4]) as a durable resource.

[4] http://thedata.org

Capturing Provenance Data

Possible approaches

How can we retrofit a provenance-tracking capability to the R interpreter?

Capturing Provenance Data

Possible approaches

If we pursue the cake-making analogy, there are (at least) two possible approaches to keeping track of provenance:

Instrument the recipes:
i.e. modify the recipes:

- to ensure that notes are kept of exactly which ingredients were used,
- to provide the resulting cake with a unique identifier, and
- to record that identifier.

Instrument the larder:
While a recipe is under preparation:

- take a note of exactly which ingredients are removed from the larder (or fridge, etc.),
- ensure that any cakes etc. newly inserted into the larder have a unique identifier, and
- associate that identifier with the current recipe invocation.

Capturing Provenance Data

Possible approaches

If we pursue the cake-making analogy, there are (at least) two possible approaches to keeping track of provenance:

Instrument the recipes:
i.e. modify the recipes:

- to ensure that notes are kept of exactly which ingredients were used,
- to provide the resulting cake with a unique identifier, and
- to record that identifier.

Instrument the larder:
While a recipe is under preparation:

- take a note of exactly which ingredients are removed from the larder (or fridge, etc.),
- ensure that any cakes etc. newly inserted into the larder have a unique identifier, and
- associate that identifier with the current recipe invocation.

Approaches to Capturing Provenance Data

Pros and cons

Instrument the recipes:

- + Instrumentation can record exactly the information of practical importance;
- + OPM roles can be clearly distinguished, and labelled in a meaningful way.
- – There are an awful lot of recipes (i.e. R functions) to instrument!

Instrument the larder:

- – Possibility of overwhelming the provenance record with dependency data of little practical significance;
- – Difficult to identify OPM roles unambiguously, and to label them meaningfully.
- + Instrumentation needs to be applied only in a few places. (In R, certain environments, including the global environment.)
- – Recipes may be able to modify the contents of the larder without ‘opening the door’ (i.e. in a way that isn’t evident to the instrumentation).

Approaches to Capturing Provenance Data
Pros and cons

Instrument the recipes:

+ Instrumentation can record exactly the information of practical importance;
+ OPM roles can be clearly distinguished, and labelled in a meaningful way.
– There are an awful lot of recipes (i.e. R functions) to instrument!

Instrument the larder:

– Possibility of overwhelming the provenance record with dependency data of little practical significance;
– Difficult to identify OPM roles unambiguously, and to label them meaningfully.
+ Instrumentation needs to be applied only in a few places. (In R, certain environments, including the global environment.)
– Recipes may be able to modify the contents of the larder without ‘opening the door’ (i.e. in a way that isn’t evident to the instrumentation).

Proposed Approach

- Instrument R functions that directly modify the contents of an R environment. This includes assignment and related functions, and functions with side effects. In other words, the OPM `wasGeneratedBy` relation is captured by ‘instrumenting the recipe’.
- Detect which objects are *read* by a top-level R command by instrumenting the global environment (and some others). In other words, the OPM `used` relation is captured by ‘instrumenting the larder’.

Proposed Approach

- Instrument R functions that directly modify the contents of an R environment. This includes assignment and related functions, and functions with side effects. In other words, the OPM `wasGeneratedBy` relation is captured by ‘instrumenting the recipe’.
- Detect which objects are *read* by a top-level R command by instrumenting the global environment (and some others). In other words, the OPM `used` relation is captured by ‘instrumenting the larder’.

Unresolved Issues and Wishlist

- How to identify and handle provenance-trusted packages.
- How to identify and handle other external resources which are trusted to be durable and maintain provenance information.
- Provision and handling of annotations.
- Handling of search-list effects in generating replay scripts.
- The interface to provenance information from within R itself.
- Tracking the provenance of graphical objects (plots, etc.).
- Shared access to the object repository?