DEEP: A Provenance-Aware Executable Document System

Huanjia Yang[1], Danius T. Michaelides[1], Chris Charlton[2], William J. Browne[3], and Luc Moreau[1]

[1] Electronics and Computer Science, University of Southampton, UK
{hy2,dtm,L.Moreau}@ecs.soton.ac.uk
[2] Graduate School of Education, University of Bristol, UK
c.charlton@bristol.ac.uk
[3] School of Veterinary Science, University of Bristol, UK
william.browne@bristol.ac.uk

Abstract. The concept of executable documents is attracting growing interest from both academics and publishers since it is a promising technology for the the dissemination of scientific results. Provenance is a kind of metadata that provides a rich description of the derivation history of data products starting from their original sources. It has been used in many different e-Science domains and has shown great potential in enabling reproducibility of scientific results. However, while both executable documents and provenance are aimed at enhancing the dissemination of scientific results, little has been done to explore the integration of both techniques. In this paper, we introduce the design and development of DEEP, an executable document environment that generates scientific results dynamically and interactively, and also records the provenance for these results in the document. In this system, provenance is exposed to users via an interface that provides them with an alternative way of navigating the executable document. In addition, we make use of the provenance to offer a document rollback facility to users and help to manage the system's dynamic resources.

1 Introduction

e-Science aims to make available complex computation and analysis to users via tools that are easy to use and understand. In the context of quantitative social science, we observe that cutting edge methodological developments are beyond the reach of some social scientists that might benefit from new and complex analysis tools. The e-Stat project brings together statisticians, social and computer scientists in a collaboration funded by the UK's Economic and Social Research Council to build an environment for social scientists that provides learning pathways to bring these cutting edge developments into their working practices.

We also observe that traditional paper-based documents come short of meeting the goals of disseminating complex scientific research and that executable

documents may provide a possible solution. In the e-Stat project, we have developed DEEP(Documents with Embedded Execution and Provenance), a system that combines document presentation with a computational back-end, thereby combining the narrative and expository advantages of conventional documents with the interactive and experimental advantages of computational methods, allowing researchers to share research findings and techniques and also document their research process. Our document reading interface allows users to explore beyond the document content and examine the dynamically generated content in detail. This facility allows readers to get a deep understanding of the computation that a DEEP document encapsulates, providing a valuable learning pathway. DEEP is part of the statistical modelling package called Stat-JR[1] developed during the course of the e-Stat project.

It is vital for DEEP to keep track of the computational processes that occur and the dynamic content that they generate. This information is essential to understand artifacts created in context and is required in order that results can be validated, reproduced and reused. This matches the principle of data provenance models, which represent the information that can help determine the nature and derivation history of a data product[2]. In DEEP, we integrate provenance generation based on a specialization of the PROV Data Model (PROV-DM)[3]. The contributions of this work are threefold: firstly, we have designed a *provenance data model* to describe the internal behaviour and the resource organization of our executable document system; secondly, the information expressed according to this data model is used to provide users with novel resource and document *navigation experience*; thirdly, provenance information is also used to drive certain system functions, such as performing *execution status checking* and *document rollback*.

The remainder of this paper is organized as follows: we survey some related work in Section 2 before extracting the requirements and presenting some basic system design principles for the e-Stat executable document system in Section 3. We discuss our integration of provenance in the internal data model of DEEP in Section 4. In Section 5, we introduce the system functions that allow rendering and navigation of provenance information. The provenance-driven system functions are presented in Section 6. Finally we conclude our work in Section 7 before discussing our future work.

2 Related work

Academic papers have always been the primary approach by which research results are disseminated within the science community. Their shortcomings, however, are also well recognized as not being able to provide sufficient support for verification, reproducibility and reuse of the research results that they describe. With the rapid developments of e-Science, the possibility of making interactive digital publications with more comprehensive information embedded within them has attracted interest from both academics and science publishers [4]. Bechhofer et al. [5] proposed the notion of Research Object (RO), which is defined

as an aggregation of essential resources and information relating to experiments and investigations that helps other people to reproduce and reuse research results. One of the key motivations of such RO notion is its potential in supporting "rich publication". Researchers and publishers who are interested in such notion gathered together in the Beyond the PDF [6] workshop, in which a variety of models, publishing tools, and impact metrics were introduced. However, most of them focus on the annotation, linked data and bundling models for static resources, with no concrete design or development for executable document. In their work on verifiable computational scientific research, Gavish and Donoho[7] introduce the notion of identifying computational results via a URL and also establishing public repository services to archive all published results. The authors argue that proper usage of this notion and service structure will simplify the practice of reproducible research and executable papers, but no solution is explicitly given to develop this claim further. The Author-Review-Execute Environment[8] has a similar notion of linked results, but it locates the results archive on authors' own machines. This requires that each author maintains a server and installs the service to expose the data and the execution resources, which raises issues of security and adoption. The SHARE environment[9] shows more progress by providing the execution services directly on its server. However, it still has not achieved an integrated interface for both the paper reading and the executions. Instead, for accessing the original data and executions, users have to use a separate view that just leads to a remote virtual machine with the required execution environment. The Collage system[10] joins the static content in the documents with interactive/dynamic components that enable the readers and reviewers to access original data contained within to validate the results by re-executing the software that generated them, and to get the document dynamically updated with the latest results. Compared to other existing systems, Collage provides a unified, dynamic and interactive document reading interface for an improved reading experience. However, it lacks the flexibility of supporting multiple executions in one document and the ability of navigating the resource structure.

Provenance is well understood in the context of art or digital libraries, where it refers to the documented history of an art work or a digital object respectively[11]. Provenance has also shown great potential in the e-Science domain, as it provides a data product's derivation history, which is crucial information for validating and reproducing the results[2]. It allows users to understand, verify and even reuse the data, and thus helps achieve a better level of research reproducibility. For the past decade, much work has been done to advocate provenance in workflow applications in various scientific domains[12], where provenance has shown some of its promising features in leveraging effective dissemination of research results. However, little has been done to integrate provenance into the field of executable documents. The authors in [13] propose a provenance based infrastructure to support the executable document's life cycle, while in [14] the authors attempt to create paper publications with provenance embedded in them to describe appropriate data and results. However, in both these papers, the pro-

posed designs depend strongly on a specific workflow system for executions and content reading.

Some systems hide the complexities of running workflows from the user by providing easy to use front-ends configured for the the application in mind. VisMashup [15] allows the creation of custom visualization applications using VisTrails as the underlying dataflow system. Web applications are a popular delivery platform such as in the Digital Synthesis Framework[16].

3 DEEP requirements and design

The requirements for DEEP documents and the DEEP system, based on the project scenarios are as follows:

Interactive: DEEP documents should provide a compelling, interactive and immersive environment. They should be reactive to user input and authors must be able to write content that can be tailored to the reader's inputs.

Interface with significant computation: DEEP should integrate with execution back-ends to perform non-trivial computation. Such integration should be seamless and maintain the document metaphor.

Exploratory: DEEP and documents written for the system should allow the reader to explore the material assembled within the document and should support them in understanding the relationships between elements of the document.

Complete access: the user should be able to view all static and dynamic resources used and generated the DEEP document and not just those those the author chose to show in the main body of the document. Such material would help improve the user's understanding, and provide a valuable resource as their capability improves.

Dynamic to static: DEEP documents have a variety of uses and we identify a spectrum of content from dynamic to static. A fully dynamic document would consist of only dynamic or computational resources - such as an electronic notebook. A static document, on the other hand, requires no computational backend as all possible dynamic resources would be contained in the DEEP document. An academic paper would be an example of such a static document. Provenance included in the document describes the relationships between any contained dynamic resources and the author would decide what dynamic resources are included.

3.1 DEEP System Design and Overview

The major components of DEEP are shown in Figure 1. A Web Browser(1) acts as the front-end providing a familiar interface to users with a strong linking

and navigation metaphor. HTML is the chosen format for the visual content in Deep, since we did not want to have to invent a new document and rendering language. In addition, by using a widely known and used format, authors should find it easier to write content (because there is a wealth of material available about writing HTML and they can use a wide range of HTML editing tools).

A significant design decision in the system is the relationship between the visual content and the execution environment. We consider a Deep document to consist of a collection of resources of different types and uses (for example static HTML content, a dataset to be used in some computation, a graph that was created by an execution). This resource-centric approach informs the design of the interface between Deep and the execution engine. The action of the user reading a Deep document establishes relevant resources which are made available to the execution engine, which, in turn, may create new resources. An "execution environment" provides the container to which resources are made available in the system. Resources have their own unique identifiers but are also "bound" into the environment with simple names ("binding names"). The unique identifiers are used by the system whereas binding names are used by the authors to anchor dynamic resources into their document as the resources become available.

The Deep Server component (4 in Figure 1) maintains the execution environments and generates notifications when resources are created, removed or bound into the execution environments. These notifications trigger activities in the browser front-end for rendering and user interaction, and in the execution engines(9) via the engine API(3) . The Deep Server uses the Resource Management component(5) for storage of Deep document files(8), provenance generation(6) and an RDF store(7) for metadata storage and querying. An HTTP server(2) exposes the Deep Server to the Web Browser and the front-end written in HTML and Javascript.

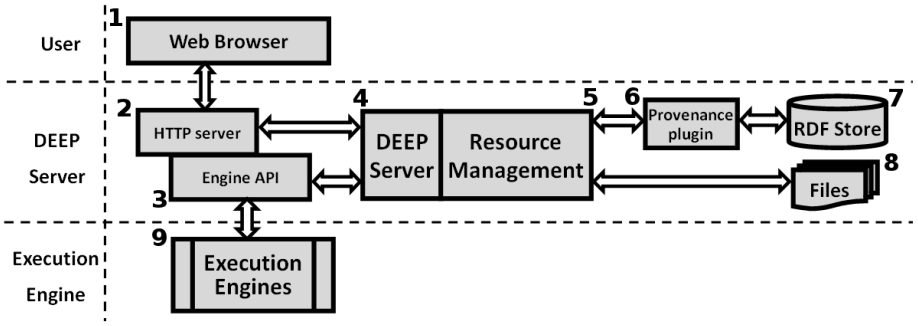

Fig. 1. Deep system structure

Figure 2 shows the browser based reading interface showing an example Deep document from the e-Stat project. The interface consists of navigational ele-

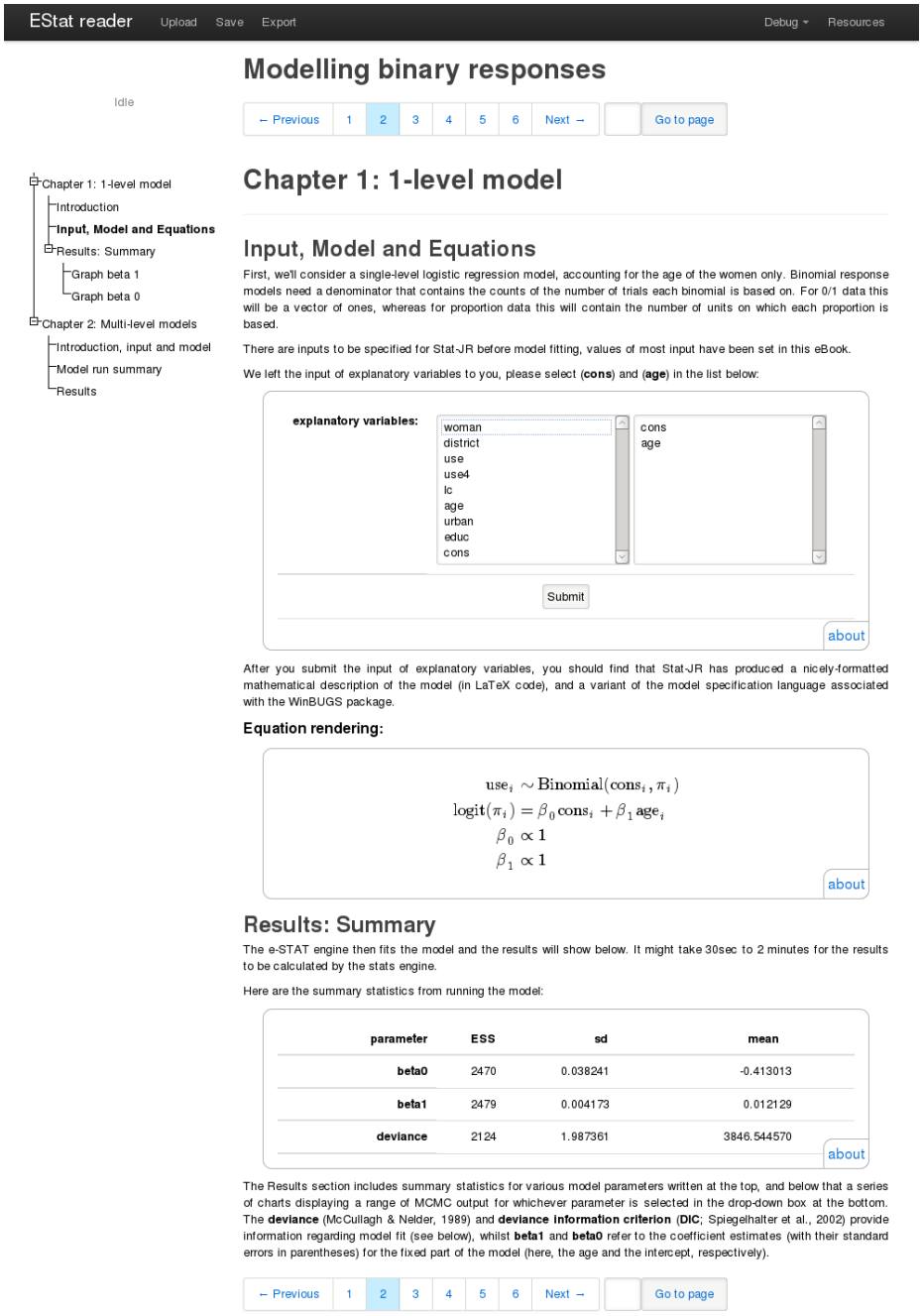

Fig. 2. Screenshot of the reader interface displaying a DEEP document

ments: the menubar at the top, the page number lists top and bottom and the contents list top to the left. The main body of the document shows a number of paragraphs of static content with dynamic content (in boxes with curved corners) placed at certain points between them. The first piece of dynamic content consists of an input widget in which the user can select the explanatory variables for the statistical model. The remaining items of dynamic content (some mathematical notation and a table) have been generated as a result of the user input. At any time, the user is free to return to the input widget and make a different selection, and so triggering a new computation, the generation of new resources and causing the document to update.

There are a range of visual behaviours available to enable the document to react to the presence of dynamic resources. In this example, when the user first begins reading, the content below the input box is hidden and is only revealed after the computation and the dynamic resources are generated. Other behaviours include hiding content, behaviours that are conditional on expressions and extracting specific fragments from resources.

Since the DEEP reading front-end is a browser, all dynamic resources must have HTML renderings but they may have other representations such as XML and CSV to enable exporting of resources and also to facilitate more complex interface widgets. Alternative rendering front-ends could be implemented by supporting appropriate representations of these dynamic resources and translating of the static content.

3.2 DEEP document structure and Reading

DEEP documents are structured in two manners: content is grouped into pages for presentation; and pages are grouped into "activity regions" for execution purposes. Activity regions have resources associated with them and these resources are only active when the reader is reading a page in that region. This structure allows authors to have a degree of control over when execution occurs and also means that DEEP documents can have many executions without the system having to instantiate all resources at once.

When a user reads a DEEP document, the system creates a number of structures to maintain state. A "reading process" is a top level container that describes the action of reading a document. Within a reading process, the action of reading an activity region is described by an "activity". An activity can contain one or more "executions". Executions are typically triggered by user input. Multiple inputs by the user result in multiple executions. The representation of these structures and the mapping to common provenance terms is discussed in the following section.

4 Provenance data model for an executable document system

On the basis of the requirements and the DEEP document file structure discussed in the previous section, we present a provenance data model to describe the system's behaviour and resource organization.

Our model is based on PROV [3], a standardization of a number of provenance vocabularies[17]. As shown in Figure 3, the internal provenance data model consists of two components: the definition component and the runtime component. The definition component consists of information defined by the author in the DEEP document file, which is loaded and stored in DEEP's RDF store when the file is imported. The information is the DEEP document's descriptive metadata, which provides the basic information for the document as well as the organization structure of the static content and resources contained within it. More specifically, it describes the activity regions contained in the document and the resources associated with each of them. All the resource files contained in the DEEP document, as well as the file itself and the activity regions contained, are considered as PROV entities, and are represented as ellipses in Figure 3. One thing to note is that an activity region is linked to each of its resources via a resource binding, which is a ternary relation that also specifies a "binding name". This allows an actual resource to be bound with multiple activity regions, but with different names in each of them to avoid confusion. It also provides a simple way for DEEP document authors to notate the resources and to place the dynamic content in the document's HTML content.

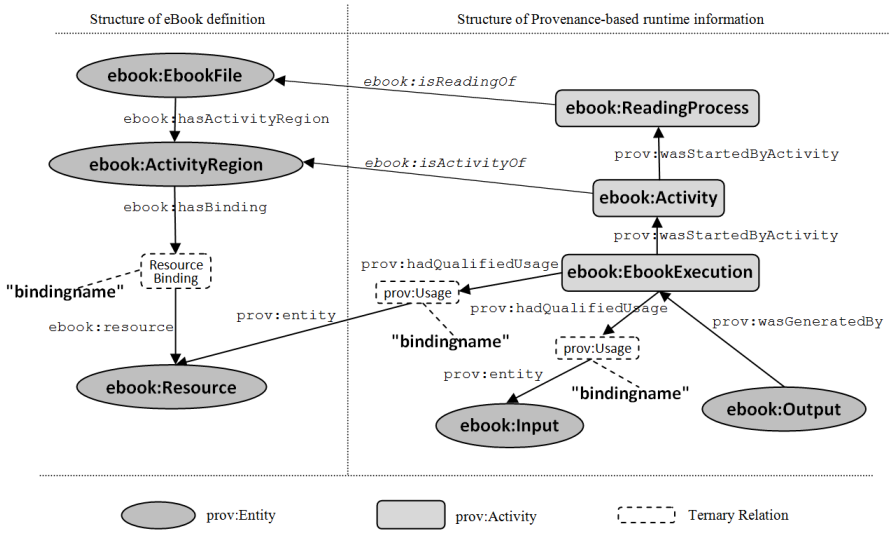

Fig. 3. Structure of data defined in DEEP document files and provenance recorded during reading

The runtime component contains the provenance information recorded automatically during DEEP document reading. For this information, three activity types are defined in an `ebook` namespace: `ReadingProcess`, `Activity` and `EbookExecution`. They are subtypes of `prov:Activity` and represent the reading of an DEEP document, the notion of being in an activity region and a specific execution respectively. The relations `isReadingOf` and `isActivityOf` associate them with appropriate static structures. The PROV `wasStartedByActivity` relation expresses that the `ReadingProcess` initiates an `Activity` when the reader enters an activity region and also that an `Activity` initiates an `EbookExecution` as a result of the execution engine having appropriate resources. All types of resource consumed by the `EbookExecution` processes are subtype of `prov:Entity`, including `ebook:Resource`, which is the resource file already defined in the definition component, and `ebook:Input`, which is the collection of parameters given by the user during reading. They are all linked to the corresponding `EbookExecution` processes with the PROV `Usage` relation. The ternary relation is used so that the "binding name" that the resource used in the execution can be specified. The results generated by `EbookExecution` are of type `ebook:Output`, a subtype of `prov:Entity`, and are linked to the corresponding `EbookExecution` processes with the PROV relation `wasGeneratedBy`.

Using this model, the system can construct a provenance graph for each reading process created by the user. It is a directed graph that grows as the user's reading activity proceeds. As we use a semantic web backend with an RDF store to persist and query data, such provenance graph is recorded with terms from the PROV ontology [18].

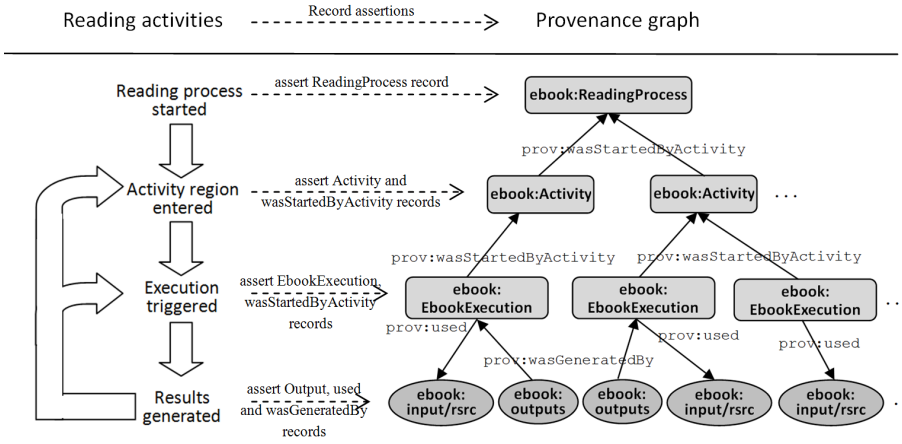

Fig. 4. DEEP document reading activities and the construction of provenance graph

The construction of the provenance graph with respect to the general system activities involved in reading a DEEP document is shown in Figure 4. The asser-

tions of the provenance record are caused by a few key events during reading. Those events include starting a new reading process, entering an activity region, triggering an execution and the generation of an execution result. The first three events are user-triggered events, which cause assertion of `prov:Activity` records with corresponding subtypes and `prov:wasStartedByActivity` records where appropriate. In the case of triggering an execution, the `EbookExecution` instances are also linked to each static resource it consumed with a `used` relation.

The last event, however, is triggered internally when a result is obtained from the execution engine. The system creates an `ebook:Output` instance for the result with a corresponding subtype depending on its nature. For example, an execution may consume statistical template, dataset and input files to generate results in the form of figures, tables, code and LaTeX based equations etc. They are represented in the provenance graph with a URI and an attribute that points to their file location. Moreover, they are directly connected to the instance of the `ebook:EbookExecution` process that generated them by the relation `prov:wasGeneratedBy`.

With the provenance data model and its implementation, the system is able to perform automated recording of the provenance graph that describes the complete system activity and dynamic resource organization within an Deep document. In the following two sections, we show that by properly presenting or extracting the required information from the provenance recorded, the system can enable additional functionality and provide the users with a better reading experience.

5 Presenting and navigating provenance information - exposing provenance to support users

Provenance information recorded in existing provenance-aware systems is usually linked to the corresponding resource, so that it can be used by itself or other applications. Whilst this information is traditionally consumed by machines, we also consider it useful in certain circumstances to expose the provenance to human users.

In Deep, dynamic resources are generated at many stages in the user's reading process. Although an author may write about these dynamic resources in the body of an document, the user may still need additional information regarding these resources and the executions that generated them for the following reasons:

- A reader may want to see an overview of the dynamic resources and the resources used to generate them, either because the document does not go into enough detail or because a clearer view will help in understanding the relationships between the various resources;
- The Deep document allows the reader to try out different inputs for the same execution. As they do so, corresponding dynamic resources embedded in the Deep document content are updated with the new results. However, the reader may want to access old results in order to make comparisons;

– Executions triggered by the reading process may generate more outputs than those that the DEEP document author has chosen to show directly in the document. The reader might be interested in these additional resources.

In order to expose this additional information, we have introduced the "resource view" into the reading interface. It is accessible from the menu bar at the top of the reading interface or by clicking on the "About" link next to each dynamic resource embedded in the document content. The resource view, shown in Figure 5, consists of a resource tree view and an information panel, which contains four tabs displaying the content, information, provenance and export links of a selected resource.

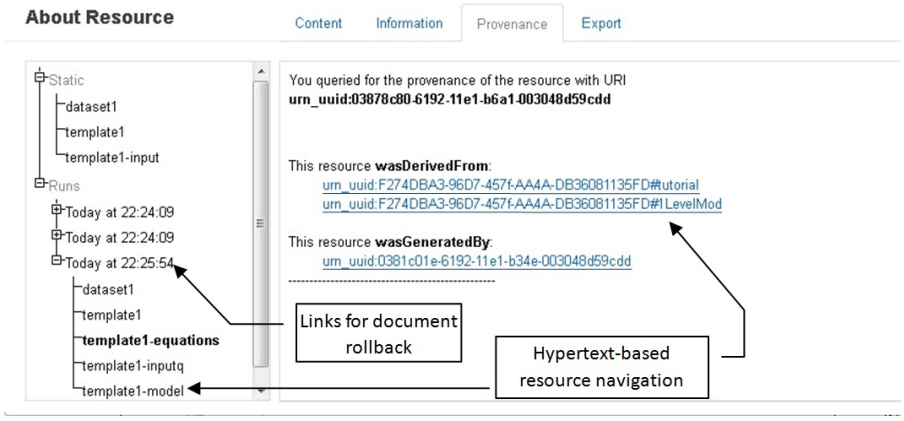

Fig. 5. The resource tree (left) and the provenance tab (right)

The resource tree shows the resources used in the current reading process in two categories: 'Static' and 'Runs'. The 'static' resources are included in the DEEP document to support the executions. In the context of the statistical application these may include statistical model templates (which construct statistical models), datasets and pre-defined input sets. The 'Runs' category lists executions that have been carried out in the current activity region of the document. Each of the executions can be further extended to show a sub-tree of all the resources used and generated allowing the user to gain a deeper understanding of the dynamic nature of the document. These executions could be those triggered by the author and subsequently included in the DEEP document or they could be executions instigated by the reader. The distinction between the two is not clear in the resource tree interface (except via the time labels) and needs improving.

In the information panel, the *Content* tab displays the actual content of the resource, whilst the *Information* tab shows additional metadata associated with it. The *Provenance* tab shows the provenance of a resource including its relationship with the reading process, executions and other resources. The *Export*

tab allows the user to extract resources from the DEEP document in appropriate formats. The resource view is designed to let the user navigate around the resources in the usual hypertext manner by clicking on links in both the resource tree and the information and provenance tabs.

The resource view is fully driven by the provenance graph of the current reading process. This graph is obtained from a provenance service in the backend server by making an HTTP GET request on the provenance URL of a reading process. When a provenance request is received, the provenance service traverses the named-graph for the reading process, and builds the provenance graph using Provpy, a Python-based binding for PROV-DM [3] that we have developed. The library serializes the provenance graph into PROV-JSON[19]. On the client side, we have developed a simple JavaScript library "provjs" to parse and query PROV-JSON graphs and the web application uses it to build the resource tree and the information and provenance tabs.

6 Provenance-aware interactive reading - using provenance to drive system functionality

Many existing provenance-aware systems focus on the ability of automatically recording and sharing of provenance information (provenance is often just recorded, and made available as raw data). In DEEP, we take this a step further by not only exposing provenance information, but also by using it to drive some system functionality: the DEEP document execution status checking and document rollback.

6.1 DEEP document execution status checking

The document execution status checking is used by DEEP to determine whether a specific computation needs to be performed. By avoiding unnecessary computation, and instead drawing on previously calculated results, DEEP is more responsive to user interaction. The presence of these reusable results arises from two situations. Firstly, they may be stored in the DEEP document file because the author determined that they were important (for example in a static document where all the possible dynamic resources have be pre-calculated). Secondly, as a user reads and interacts with a DEEP document dynamic resources are created. If the user returns to a configuration of inputs that has been explored before, the dynamic resources generated previously can be reused.

Given the resource-centric design of DEEP and execution environment, the query to determine whether a previous execution can be reused is simply stated as: given the current set of inputs, is there an existing execution that has exactly the same set of inputs with exactly the same mapping to bound names. Figure 6 shows the SPARQL query that we generate to determine suitable executions where we have an `prov:Used` for each input and the variables `cur_act_region` and `input1` to `inputN` are passed in as parameters to the query. This query is executed on the named-graph for the current reading process and if there is

a matching execution, the system finds all the outputs for that execution and reinstates those resources. Essentially, the combination of resource storage and our provenance information allow us to perform a form of memorization where the focus is on the needs of the document and is agnostic to the execution engine. This caching, however, is similar to caching that occurs in some workflow systems such as the VisTrails Cache Manager[20] in the VisTrails system. Execution status checking is not a frequently triggered activity, so, although performance of the SPARQL queries has not been observed to be critical, the technique of finding existing executions by querying using a signature of inputs could be applied here should the provenance graphs become large (i.e. large numbers of executions or executions with large numbers of inputs).

```
SELECT ?exec WHERE { ?exec rdf:type ebook:EbookExecution.
        ?exec prov:wasStartedByActivity ?activity.
        ?activity ebook:isActivityOf ?cur_act_region.
        ?exec prov:used ?input1.
        ?exec prov:used ?input2.
        ...
        ?exec prov:used ?inputN. }
```

Fig. 6. The SPARQL query used to check for an existing execution

6.2 DEEP document rollback

The other system function driven by provenance is document rollback. During the reading of a DEEP document, the user can return to parameter input areas in the document and give different responses. This will trigger new executions and cause the document content to update with different results. Although the resource view allows the inspection of individual results from any execution, the user may prefer to see them in the context of the DEEP document content and therefore wish to revisit a previous execution by returning the state of the document to that point in time. This is performed purely from the point of view of the document and document reading infrastructure and is not reliant on support from the execution engine.

We expose this document rollback facility by allowing the user to click on executions in the resource tree shown in Figure 5. In response, DEEP must reinstate all resources that were generated by the relevant execution. These resources are found by querying the provenance record with a SPARQL query and rebinding them into the execution environment.

These two facilities of DEEP rely on the recording of provenance and the use of RDF and SPARQL to represent and query it.

7 Conclusion and Future work

Executable documents have become a promising e-Science technology that aims to increase comprehension and reproducibility in the dissemination of scientific results. Data provenance has also shown its potential to enable reproducibility of research results by providing a uniform description of their derivation history. In an attempt to bring together these two technologies, DEEP integrates provenance in an executable document system. This paper reported three main contributions in terms of provenance study. Firstly, we have integrated provenance with the system's internal data structure by using a specialization of the PROV data model to describe the behaviour and resource organization of the system. By recording this provenance for all the dynamic results generated during reading, DEEP is able to provide their full derivation history. Secondly, in terms of the usage of provenance, we have shown that, in our interface, provenance can be exposed to and navigated by DEEP users. This provides the users with a different level of understanding of the resource structure, as well as new ways to navigate the document. Thirdly, we have designed two of DEEP's features, the execution status checking and the document rollback, to be based fully on provenance. This demonstrates that data provenance is not just information to be shown to the reader, but can also be used to drive the system functionality.

DEEP provides a framework that could be applicable to a broad range of scientific domains. With the integration of provenance in our DEEP documents, we can tackle issues of verification and reproducibility and our future work will aim to improve DEEP's infrastructure and functionality to support this. Such work could be twofold: firstly, we aim to make provenance exportable and transferable within the DEEP document files. This means that a user could generate their results in the form of an document and disseminate it to other readers. The provenance carried with the document will help the reader to understand, examine and even reproduce results for the purposes of validating or reusing them. Secondly, our use of a unified representation of provenance, means that we could integrate with other provenance-aware software and provide the user with more detailed provenance, allowing reproducibility and validation at various levels. In terms of modelling and implementation, both of those two points may lead to the notion of provenance accounts being introduced into our system. This would allow us to bundle provenance generated by different users so that DEEP documents can be distributed multiple times to support collaborative work, and, to bundle provenance from different components, including third party software, for better information granularity.

Acknowledgments

This researched was conducted as part of the E-Stat project, funded by the ESRC (RES-149-25-1084) under the Digital Social Research programme. We wish to thank Richard Parker and our other colleagues at the Centre for Multilevel Modelling for their input into the design of our system.

References

1. The eStat Project: Stat-JR. http://www.bristol.ac.uk/cmm/research/estat/
2. Simmhan, Y.L., Plale, B., Gannon, D.: A survey of data provenance in e-science. SIGMOD Rec. **34**(3) (September 2005) 31–36
3. Moreau, L., Missier, P.: The PROV Data Model and Abstract Syntax Notation. http://www.w3.org/TR/prov-dm/ Retrieved 28th March 2012.
4. de Waard, A.: The Future of the Journal? Integrating research data with scientific discourse. Nature Precedings (713)
5. Bechhofer, S., Buchan, I., Roure, D.D., Missier, P., Ainsworth, J., Bhagat, J., Couch, P., Cruickshank, D., Delderfield, M., Dunlop, I., Gamble, M., Michaelides, D., Owen, S., Newman, D., Sufi, S., Goble, C.: Why linked data is not enough for scientists. Future Generation Computer Systems (2011)
6. Bourne, P., de Waard, A.: Beyond the PDF Workshop. http://sites.google.com/site/beyondthepdf (2011)
7. Gavish, M., Donoho, D.: A Universal Identifier for Computational Results. Procedia Computer Science **4** (January 2011) 637–647
8. Müller, W., Rojas, I., Eberhart, A., Haase, P., Schmidt, M.: A-R-E: The Author-Review-Execute Environment. Procedia Computer Science **4**(0) (2011) 627–636
9. Gorp, P.V., Mazanek, S.: SHARE: a web portal for creating and sharing executable research papers. Procedia Computer Science **4**(0) (2011) 589–597
10. Nowakowski, P., Ciepiela, E., Harężlak, D., Kocot, J., Kasztelnik, M., Bartyński, T., Meizner, J., Dyk, G., Malawski, M.: The Collage Authoring Environment. Procedia Computer Science **4** (January 2011) 608–617
11. PREMIS Working Group: Data dictionary for preservation metadata. Technical report (2005)
12. Moreau, L.: The Foundations for Provenance on the Web. Found. Trends Web Sci. **2**(2–3) (February 2010) 99–241
13. Koop, D., Santos, E., Mates, P., Vo, H.T., Bonnet, P., Bauer, B., Surer, B., Troyer, M., Williams, D.N., Tohline, J.E., Freire, J., Silva, C.T.: A Provenance-Based Infrastructure to Support the Life Cycle of Executable Papers. Procedia Computer Science **4**(0) (2011) 648–657
14. Bauer, B., Gukelberger, J., Surer, B., Troyer, M.: Publishing provenance-rich scientific papers. Procs TAPP11 Theory and Practice of Provenance (2011)
15. Santos, E., Lins, L.D., Ahrens, J.P., Freire, J., Silva, C.T.: VisMashup: Streamlining the Creation of Custom Visualization Applications. IEEE Trans. Vis. Comput. Graph. **15**(6) (2009) 1539–1546
16. Myers, J., Marini, L., Kooper, R., McLaren, T., McGrath, R.E., Futrelle, J., Bajcsy, P., Collier, A., Liu, Y., Hampton, S.: A Digital Synthesis Framework for Virtual Observatories, Edinburgh, UK (2008)
17. Sahoo, S., Groth, P., Hartig, O., Miles, S., Coppens, S., Myers, J., Gil, Y., Moreau, L., Zhao, J., Panzer, M., Garijo, D.: Provenance Vocabulary Mappings. Technical report, W3C Provenance Incubator Group (August 2010)
18. Sahoo, S., McGuinness, D.: The PROV Ontology: Model and Formal Semantics. http://www.w3.org/TR/prov-o/
19. Huynh, T., Jewell, M., Keshavarz, A., Michaelides, D., Moreau, L., Yang, H.: The PROV-JSON Serialization. http://users.ecs.soton.ac.uk/tdh/json/
20. Bavoil, L., Callahan, S., Crossno, P., Freire, J., Scheidegger, C., Silva, C., Vo, H.: Vistrails: enabling interactive multiple-view visualizations. In: Visualization, 2005. VIS 05. IEEE. (oct. 2005) 135 – 142