

Luc Moreau
Ian Foster (Eds.)

LNCS 4145

Provenance and Annotation of Data

International Provenance and Annotation Workshop, IPAW 2006
Chicago, IL, USA, May 2006
Revised Selected Papers

 Springer

Commenced Publication in 1973

Founding and Former Series Editors:

Gerhard Goos, Juris Hartmanis, and Jan van Leeuwen

Editorial Board

David Hutchison

Lancaster University, UK

Takeo Kanade

Carnegie Mellon University, Pittsburgh, PA, USA

Josef Kittler

University of Surrey, Guildford, UK

Jon M. Kleinberg

Cornell University, Ithaca, NY, USA

Friedemann Mattern

ETH Zurich, Switzerland

John C. Mitchell

Stanford University, CA, USA

Moni Naor

Weizmann Institute of Science, Rehovot, Israel

Oscar Nierstrasz

University of Bern, Switzerland

C. Pandu Rangan

Indian Institute of Technology, Madras, India

Bernhard Steffen

University of Dortmund, Germany

Madhu Sudan

Massachusetts Institute of Technology, MA, USA

Demetri Terzopoulos

University of California, Los Angeles, CA, USA

Doug Tygar

University of California, Berkeley, CA, USA

Moshe Y. Vardi

Rice University, Houston, TX, USA

Gerhard Weikum

Max-Planck Institute of Computer Science, Saarbruecken, Germany

Luc Moreau Ian Foster (Eds.)

Provenance and Annotation of Data

International Provenance and Annotation Workshop
IPAW 2006
Chicago, IL, USA, May 3-5, 2006
Revised Selected Papers



Springer

Volume Editors

Luc Moreau
University of Southampton
Southampton, UK
E-mail: l.moreau@ecs.soton.ac.uk

Ian Foster
Argonne National Lab
University of Chicago
Chicago, U.S.A.
E-mail: foster@mcs.anl.gov

Library of Congress Control Number: 2006933370

CR Subject Classification (1998): H.3, H.4, D.4, E.2, H.5, K.6, K.4

LNCS Sublibrary: SL 3 – Information Systems and Application, incl. Internet/Web and HCI

ISSN 0302-9743
ISBN-10 3-540-46302-X Springer Berlin Heidelberg New York
ISBN-13 978-3-540-46302-3 Springer Berlin Heidelberg New York

This work is subject to copyright. All rights are reserved, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, re-use of illustrations, recitation, broadcasting, reproduction on microfilms or in any other way, and storage in data banks. Duplication of this publication or parts thereof is permitted only under the provisions of the German Copyright Law of September 9, 1965, in its current version, and permission for use must always be obtained from Springer. Violations are liable to prosecution under the German Copyright Law.

Springer is a part of Springer Science+Business Media
springer.com

© Springer-Verlag Berlin Heidelberg 2006
Printed in Germany

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India
Printed on acid-free paper SPIN: 11890850 06/3142 5 4 3 2 1 0

Preface

Provenance is a well understood concept in the study of fine art, where it refers to the documented history of an art object. Given that documented history, the object attains an authority that allows scholars to understand and appreciate its importance and context relative to other works. In the absence of such history, art objects may be treated with some skepticism by those who study and view them.

Over the last few years, a number of teams have been applying this concept of provenance to data and information generated within computer systems. If the provenance of data produced by computer systems can be determined as it can for some works of art, then users will be able to understand (for example) how documents were assembled, how simulation results were determined, and how financial analyses were carried out.

A key driver for this research has been e-Science. Reproducibility of results and documentation of method have always been important concerns in science, and today scientists of many fields (such as bioinformatics, medical research, chemistry, and physics) see provenance as a mechanism that can help repeat scientific experiments, verify results, and reproduce data products. Likewise, provenance offers opportunities for the business world, since it allows for the analysis of processes that led to results, for instance to check they are well-behaved or satisfy constraints; hence, provenance offers the means to check compliance of processes, on the basis of their actual execution. Indeed, increasing regulation of many industries (for example, financial services) means that provenance recording is becoming a legal requirement.

Annotation is closely related to provenance. End users do more than produce and consume data: they comment on data and refer to it and to the results of queries upon it. Annotation is therefore an important aspect of communication. One user may want to highlight a point in data space for another to investigate further. They may wish to annotate the result of a query such that similar queries show the annotation. Such annotations then become valuable because they may also provide information about the origin of data. At the same time, we may wish to understand the provenance of annotations, which may be produced by people or programs. Hence, provenance and annotation offer complementary information that can help end-users understand how and why data products were derived.

The International Provenance and Annotation Workshop (IPAW 2006) was a follow-up to workshops in Chicago in October 2002 and in Edinburgh in December 2003. It brought together computer scientists and domain scientists with a common interest in issues of data provenance, process documentation, data derivation, and data annotation. IPAW 2006 was held on May 3-5, 2006 at the

University of Chicago's Gleacher Center in downtown Chicago and was attended by roughly 45 participants.

We received 33 high quality submissions in response to the call for papers. All submissions were reviewed by 3 reviewers. Overall, 26 papers were accepted for oral presentation: 4 papers, which came with consistent high ranking by reviewers, were awarded longer presentation time and space in the proceedings; 4 papers had shorter presentations, whereas the remain 18 papers were regular. In addition, Juliana Freire (University of Utah) and Roger Barga (Microsoft Research) were invited to make keynote addresses.

The workshop was organized as a single-track event, interleaving formal presentation and discussions. It also included an entertaining "Gong Show" of outlandish, "outside-the-box" ideas. Slides and presentation materials can be found at <http://www.ipaw.info/ipaw06/programme.html>.

In a discussion session, the participants debated whether the time was right to propose standard models of provenance or standard interfaces for recording, querying, and administering provenance stores. An outcome of this discussion was that participants agreed on steps to set up a "Provenance Challenge," to include a provenance-tracking scenario and query-based evaluation; this challenge will enable different groups to test and compare their approaches for this scenario (cf. <http://twiki.ipaw.info/bin/view/Challenge>).

Overall this was a very successful event, with much enthusiastic discussion, many new ideas and insights generated, and a good community spirit. It was agreed that we should hold another of these events in about 18 months' time.

June 2006

Luc Moreau and Ian Foster

Organization

IPAW 2006 was organized by the University of Chicago, Argonne National Laboratory, and the University of Southampton.

Program Committee

Dave Berry, National e-Science Centre, UK

Peter Buneman, University of Edinburgh, UK

Ian Foster (co-chair), Argonne National Lab/University of Chicago, USA

James Frew, University of California, USA

Jim Hendler, University of Maryland, USA

Carole Goble, University of Manchester, UK

Reagan Moore, San Diego Supercomputer Center, USA

Luc Moreau (co-chair), University of Southampton, UK

Jim Myers, National Center for Supercomputing Applications, USA

York Sure, University of Karlsruhe, Germany

Ziga Turk, University of Ljubljana, Slovenia

Mike Wilde, Argonne National Lab/University of Chicago, USA

Hai Zhuge, Institute of Computing Technology, Chinese Academy of Sciences,
China

Sponsors

IPAW 2006 was sponsored by Springer and Microsoft, and endorsed by the Global Grid Forum.

Table of Contents

Session 1: Keynotes

Automatic Generation of Workflow Provenance	1
<i>Roger S. Barga, Luciano A. Digiampietri</i>	
Managing Rapidly-Evolving Scientific Workflows	10
<i>Juliana Freire, Cláudio T. Silva, Steven P. Callahan, Emanuele Santos, Carlos E. Scheidegger, Huy T. Vo</i>	

Session 2: Applications

Virtual Logbooks and Collaboration in Science and Software Development	19
<i>Dimitri Bourilkov, Vaibhav Khandelwal, Archis Kulkarni, Sanket Totala</i>	
Applying Provenance in Distributed Organ Transplant Management	28
<i>Sergio Álvarez, Javier Vázquez-Salceda, Tamás Kifor, László Z. Varga, Steven Willmott</i>	
Provenance Implementation in a Scientific Simulation Environment	37
<i>Guy K. Kloss, Andreas Schreiber</i>	
Towards Low Overhead Provenance Tracking in Near Real-Time Stream Filtering	46
<i>Nithya N. Vijayakumar, Beth Plale</i>	
Enabling Provenance on Large Scale e-Science Applications	55
<i>Miguel Branco, Luc Moreau</i>	

Session 3: Discussion

Session 4: Semantics 1

Harvesting RDF Triples	64
<i>Joe Futrelle</i>	
Mapping Physical Formats to Logical Models to Extract Data and Metadata: The Defuddle Parsing Engine	73
<i>Tara D. Talbott, Karen L. Schuchardt, Eric G. Stephan, James D. Myers</i>	

Annotation and Provenance Tracking in Semantic Web
 Photo Libraries 82
*Christian Halaschek-Wiener, Jennifer Golbeck, Andrew Schain,
 Michael Grove, Bijan Parsia, Jim Hendler*

Metadata Catalogs with Semantic Representations 90
Yolanda Gil, Varun Ratnakar, Ewa Deelman

Combining Provenance with Trust in Social Networks for Semantic
 Web Content Filtering 101
Jennifer Golbeck

Session 5: Workflow

Recording Actor State in Scientific Workflows 109
Ian Wootten, Omer Rana, Shrija Rajbhandari

Provenance Collection Support in the Kepler Scientific
 Workflow System 118
Ilkay Altintas, Oscar Barney, Efrat Jaeger-Frank

A Model for User-Oriented Data Provenance in Pipelined Scientific
 Workflows 133
*Shaun Bowers, Timothy McPhillips, Bertram Ludäscher,
 Shirley Cohen, Susan B. Davidson*

Applying the Virtual Data Provenance Model 148
Yong Zhao, Michael Wilde, Ian Foster

Session 6: Models of Provenance, Annotations and Processes

A Provenance Model for Manually Curated Data 162
*Peter Buneman, Adriane Chapman, James Cheney,
 Stijn Vansummeren*

Issues in Automatic Provenance Collection 171
*Uri Braun, Simson Garfinkel, David A. Holland,
 Kiran-Kumar Muniswamy-Reddy, Margo I. Seltzer*

Electronically Querying for the Provenance of Entities 184
Simon Miles

AstroDAS: Sharing Assertions Across Astronomy Catalogues Through Distributed Annotation	193
<i>Rajendra Bose, Robert G. Mann, Diego Prina-Ricotti</i>	

Session 7: Gong Show

Session 8: Systems

Security Issues in a SOA-Based Provenance System	203
<i>Victor Tan, Paul Groth, Simon Miles, Sheng Jiang, Steve Munroe, Sofia Tsasakou, Luc Moreau</i>	
Implementing a Secure Annotation Service	212
<i>Imran Khan, Ronald Schroeter, Jane Hunter</i>	
Performance Evaluation of the Karma Provenance Framework for Scientific Workflows	222
<i>Yogesh L. Simmhan, Beth Plale, Dennis Gannon, Suresh Marru</i>	
Exploring Provenance in a Distributed Job Execution System	237
<i>Christine F. Reilly, Jeffrey F. Naughton</i>	
gLite Job Provenance	246
<i>František Dvořák, Daniel Kourřil, Aleř Křenek, Luděk Matyska, Miloř Mulač, Jan Pospřřil, Miroslav Ruda, Zdeněk Salvet, Jiřř Sitera, Michal Vocř</i>	

Session 9: Semantics 2

An Identity Crisis in the Life Sciences	254
<i>Jun Zhao, Carole Goble, Robert Stevens</i>	
CombeChem: A Case Study in Provenance and Annotation Using the Semantic Web	270
<i>Jeremy Frey, David De Roure, Kieron Taylor, Jonathan Essex, Hugo Mills, Ed Zaluska</i>	
Principles of High Quality Documentation for Provenance: A Philosophical Discussion	278
<i>Paul Groth, Simon Miles, Steve Munroe</i>	

Session 10: Final Discussion

Author Index	287
------------------------	-----

Automatic Generation of Workflow Provenance

Roger S. Barga¹ and Luciano A. Digiampietri²

¹ Microsoft Research, One Microsoft Way
Redmond, WA 98052, USA

² Institute of Computing, University of Campinas,
Sao Paolo, Brazil
barga@microsoft.com

Abstract. While workflow is playing an increasingly important role in e-Science, current systems lack support for the collection of provenance data. We argue that workflow provenance data should be automatically generated by the enactment engine and managed over time by an underlying storage service. We briefly describe our layered model for workflow execution provenance, which allows navigation from the conceptual model of an experiment to instance data collected during a specific experiment run, and back.

1 Introduction

Many scientific disciplines today are data and information driven, and new scientific knowledge is often obtained by scientists scheduling data intensive computing tasks across an ever-changing set of computing resources. Scientific workflows represent the logical culmination of this trend. They provide the necessary abstractions that enable effective usage of computational resources, and the development of robust problem-solving environments that marshal both data and computing resources. Part of the established scientific method is to create a record of the origin of a result, how it was obtained, experimental methods used, the machines, calibrations and parameter settings, quantities, etc. It is the same with scientific workflows, except here the result provenance is a record of workflow activities invoked, services and databases used, their versions and parameter settings, data sets used or generated and so forth. Without this provenance data, the results of a scientific workflow are of limited value.

Though the value of result provenance is widely recognized, today most workflow management systems generate only limited provenance data. Consequently, this places the burden on the user to manually record provenance data in order to make an experiment or result explainable. Once the experiment is finished, the specific steps leading to the result are often stored away in a private file or notebook. However, weeks or months may pass before the scientist realizes a result was significant, by which time provenance has been forgotten or lost – it simply slips through the cracks.

We believe the collection of provenance data should be automatic, and the resulting provenance data should be managed by the underlying system. Moreover, a robust provenance trace should support multiple levels of representations. In some cases, it is suitable to provide an abstract description of the scientific process that took place, without describing specific codes executed, the data sets or remote services

invoked. In other cases, a precise description of the execution of the workflow, with all operational details such as parameters and machine configurations provided, is exactly what is required to explain a result. Given no single representation can satisfy all provenance queries, the system should generate multiple related representations simultaneously. The ideal model will allow users to navigate from the abstract levels into lower levels and map the latter backwards to higher level abstractions. This also allows scientists to specify exactly what they wish to share from their experiment.

In this paper, we suggest the workflow enactment engine should be responsible for generating workflow provenance *automatically* during runtime, and an underlying storage service should be responsible for managing workflow provenance data. Provenance collection and management should happen transparently. That is, users should not have to take any special actions or execute special commands to have provenance of their workflow execution collected and maintained. Indeed, unless users take extraordinary action, such as purging an experiment result, an accurate provenance record will be maintained for results residing on their system. By making the enactment system responsible for the creation and collection of execution provenance, we are able to generate provenance automatically, freeing users from having to manually track provenance during execution. Equally important, when properly collected and maintained, workflow provenance data can be indexed and queried as a *first class data product* using conventional data management tools and techniques. In this paper, we also outline a *layered* or *factored* model for representing workflow execution provenance, from an abstract description of the workflow (scientific process) to a precise description of execution level details for a single experiment, which enables provenance data to be linked and defined in a way for more effective discovery, integration and cooperation.

2 Workflow Execution Provenance Model

In this section we outline the machine-readable model we propose to represent provenance data captured during workflow execution. The attractions of creating a machine-readable model of an experiment, that is, a workflow, are not difficult to envisage. Three key benefits that a *multilayered model* can bring are explanation, validation, and insight. Having an abstract model of the science being undertaken, and being able to use the model to interpret behavior (services and data sets used), together make it possible to explain or reason about the science. For validation, knowing which individual activities were executed, identifying branches taken during execution, steps skipped or inserted, and specific parameters supplied at runtime is essential when trying to establish trust in a result. The ability to compare a set of related experiment executions against an abstract workflow model, to identify common patterns or options not yet explored, can be used to gain insight in authoring new experiment designs. A related benefit, not discussed in this short paper is optimization – a layered or *factored* model can expose opportunities to efficiently store provenances traces in the underlying storage manager.

The first level in our model, illustrated in Figure 1 below, represents an abstract description of the scientific process. This layer captures *abstract activities* in the workflow and *links/relationships* among the activities or ports of activities, where an

abstract activity is a *class* of activities. These classes can be described by the function of the activity (e.g., `InvokeWebService`, or `Sequential`, `Parallel Branch` and `While` activities, etc) or through semantic description – for example, a class of *alignment* activities, where `Blastx` and `FASTA` are instances that belong to this class.

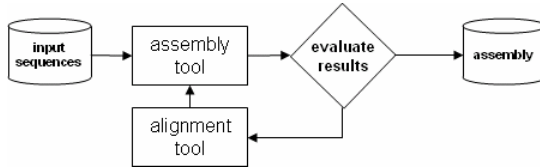


Fig. 1. Level L0, represents abstract service descriptions

The second level in our model, illustrated in Figure 2 below, represents an instance of the abstract model. This layer captures bindings or *instances of activities* and additional *relationships*, as classes of activities are instantiated. This level refines abstract activities specified in the previous level with a specific instance (for example, `InvokeWebService` is bound to a specific activity instance `InvokeBlastWebService` and `Alignment tool` is bound to `Blastx tool`). The additional information captured at this level is required to assign *fields of the classes* of the previous level. For example, the `InvokeWebService` has fields `ProxyClass` and `MethodName` that should be filled, with a proxy where the web service can be found and name of the method to be invoked.

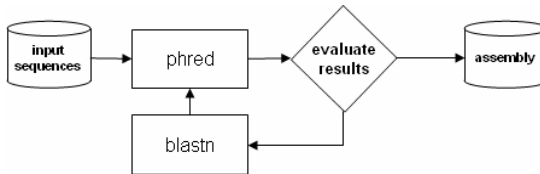


Fig. 2. Level L1, represents service instantiation descriptions

In the next level of the model L2, illustrated in Figure 3, represents the execution of the workflow instance specified in level L1. From level L1 to level L2 the model captures information provided at runtime, such as *parameters* and properties that complete the specification of activities, including input data, runtime parameter supplied, activities inserted or skipped during execution, etc. Information captured at

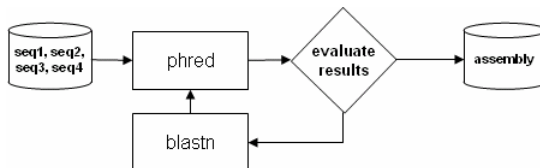


Fig. 3. Level L2, represents data instantiation of the executable workflow

this level is sufficient to trace the execution of the workflow, from input parameters to individual activities executed and runtime parameters supplied to each activity.

The final level of our model L3, illustrated in Figure 4 below, represents runtime specific information. Here we capture operational specific details, such as the start and end time of the workflow execution, the start and end time of the individual activity execution, status codes and intermediary results, information about the internal state of each activity, along with information about the machines to which each activity was assigned for execution.

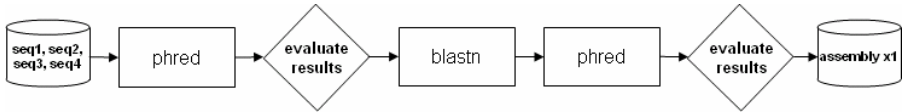


Fig. 4. Level L3, run-time provenance for a workflow execution

At each level of our model, we extend the amount of detail and amount of information of the previous level. The linkage between levels is facilitated by three types of operations: *extend* (class extension); *instance* (class instantiation) and *import* (reference to a specific model). Resuming, in level 0 the user designs a class from a model that is extended in level 1. In the level 2, the class of level 1 is instantiated. Level 3 imports the model of level 2 to make references to its entities.

3 Basic Workflow Execution Provenance Sample

In this section we illustrate the information that can be captured in our model using a simple example. Our implementation is based on the Windows Workflow Foundation

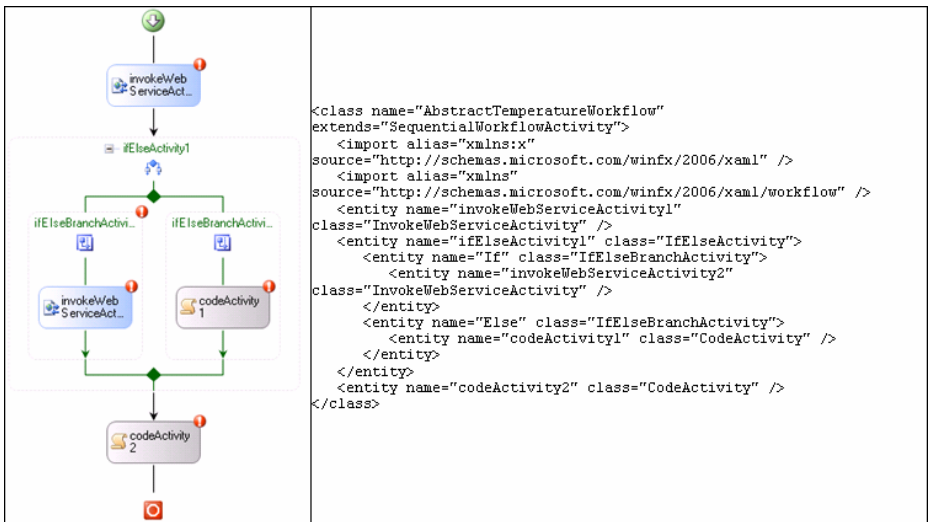


Fig. 5. Workflow level L0

(WinFX) [1], an extensible framework for developing workflow solutions. WinFX has a predefined set of activities (If-Else, While, Parallel, InvokeWebService, etc) and allows for user-defined custom activities by object inheritance from base classes.

In level L0 the user defines type of workflow (*Sequential*, *StateMachine* or *Rule Driven*) she wants to construct and identifies classes of activities available for use. Fig. 5 shows the graphical representation of the workflow and corresponding code. The data stored in the provenance for this level supports general queries, such as: what experiments in my collection use web services? (*invokeWebServiceActivities*) or what experiments utilize alignment tools? This level, in general, describes the design space for all workflows that are an instance of this abstract model.

In level L1 the user replaces abstract activities with concrete activities, and enters fields pre-defined for the individual abstract activities. For example, the activity *invokeWebServiceActivity1* of Figure 5 was specialized to *captureTemperatureWS* in Fig. 6 and the property *ProxyClass* was filled with the desired proxy. Another property that was filled in was *MethodName*, with the name of the method from the Web Service the user wants to invoke. The provenance model of L2 models a specific instance of the abstract workflow model that is being prepared for execution.

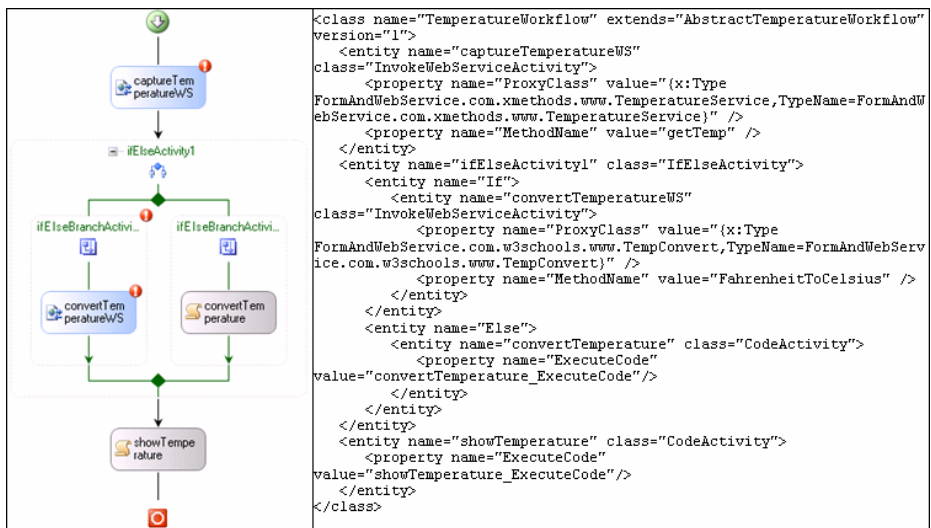


Fig. 6. Workflow level L1

From the provenance in Level L1 the user can pose queries about specific activities: What experiments (workflows) used the web service *TemperatureService*? Or, what experiments invoke the following method of a service, or what experiments use the same alignment tool?

In the level L2, which is the executable workflow, all data sets are identified and data parameters are supplied so that the workflow is ready to be executed. One example of parameter that was filled is *zipcode* from *captureTemperatureWS* that has the value “98007” (see more details in Fig. 7). In this level of the model, we can

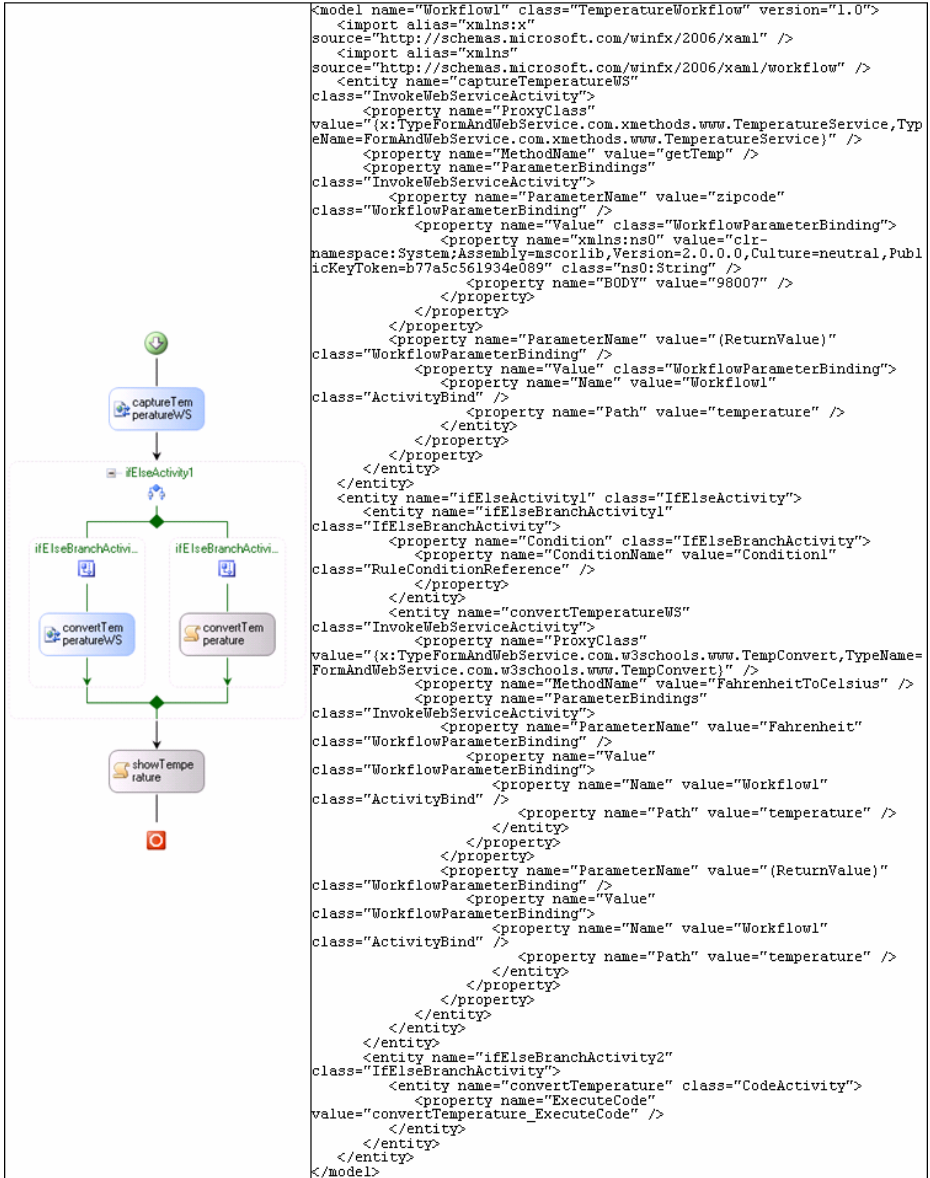


Fig. 7. Workflow level L2

pose a number of queries which do not involve runtime information. These queries can involve parameters, activities and abstract activities/structures. What workflows used the value “98007” in *zipcode* parameters? What was the precondition for the workflow follow the path *If* in the *IfThenElseActivity1*?

The last level (runtime level) can contain a great diversity of information. The kind of information to be stored will depend of the goals of the user. For example, in a system that utilizes a high performance Grid, the user may wish to know what activity (job) was executed in each processor of the Grid and when this execution occurred. In other systems, the user may wish to store information about the internal status of each activity. These two examples are allowed by our provenance model, but require changes in the applications to work (in the Grid example, the scheduler must send information about the process to the provenance system, and in the second example each activity must send information to the provenance system).

Fig. 8 shows a simple runtime record from the Executable Workflow of Fig. 7. In this example, only information about start and end time of the workflow and of the activities were stored and information about produced data.



Fig. 8. Workflow level 3

Note that in workflows designed using the Windows Workflow Foundation system there are no ports/channels to data exchange. This exchange is made via workflow global variables. Our provenance model allows for this information (and information about the code of each *CodeActivity*) be stored inside the workflow model (in level L2). The rules of the workflow also can be stored in the *ExecutableWorkflow* level. Fig. 9 shows the Code and Rules for the workflow *ExecutableWorkflow* in Fig. 7. Together, these are sufficient to recreate the original workflow specification.

required to validate a result. A multilevel representation would permit scientists to specify exactly what they wish to share, or retain, from their experiment record.

The work presented here is very much an initial investigation in an area of growing importance for e-Science. Our layered provenance model for workflow execution is a prototype to be used as a tool to explore user requirements, and to consider system requirements for managing provenance data. We envisage this provenance data to be a *first class* data resource in its own right, allowing users to both query experiment holdings to establish trust in a result and to drive the creation of future experiments.

5 Related Work

The challenge of workflow execution provenance is growing in both importance and as a research topic. Most existing provenance systems available today can be classified into one of two broad categories: i) Flow based provenance systems and ii) Annotation based provenance systems.

Flow Based Provenance Systems: In many existing Grid/Service workflow based systems, the provenance recorded is generally a variant of the following: when a service is invoked, the workflow manager stores the input, output and the service name in a provenance repository. Also, logically related service invocations are grouped by using a common ID. One disadvantage with this schema is that it can be sub-optimal for running provenance based queries, and precludes the ability to compress or reuse provenance data records. And, if not properly designed it can double the depth of the provenance trace, as traversals first have to lookup the process, then lookup inputs to the process to get to the parent file. Systems that have a workflow based provenance include: PASOA [2], Chimera [3], and myGrid [4].

Annotation Based Provenance Systems: Systems such as the storage resource broker (SRB) [5] and GeoDise [6] store provenance as name-value attribute pairs. While this schema can be used to build a provenance trace, the construction again can be very inefficient for the same reasons as in workflow based provenance systems, and it does not lend itself to a layered model.

References

- [1] Windows Workflow Foundation (WinFX), <http://msdn.microsoft.com/workflow/>.
- [2] PASOA, <http://twiki.pasoa.ecs.soton.ac.uk/bin/view/PASOA/WebHome>.
- [3] I. Foster, J. Voeckler, M. Wilde, and Y. Zhao. The Virtual Data Grid: A New Model and Architecture for Data-Intensive Collaboration. In CIDR, January. 2003.
- [4] J. Zhao, M. Goble, C. Greenwood, C. Wroe, and R. Stevens. Annotating, linking and browsing provenance logs for e-science.
- [5] M. Wan, A. Rajasekar, and W. Schroeder. Overview of the SRB 3.0: the Federated MCAT. <http://www.npaci.edu/DICE/SRB/FedMcat.html>, September 2003.
- [6] S. Cox, Z. Jiao, and J. Wason. Data management services for engineering. 2002.

Managing Rapidly-Evolving Scientific Workflows

Juliana Freire, Cláudio T. Silva, Steven P. Callahan,
Emanuele Santos, Carlos E. Scheidegger, and Huy T. Vo

University of Utah

Abstract. We give an overview of VisTrails, a system that provides an infrastructure for systematically capturing detailed provenance and streamlining the data exploration process. A key feature that sets VisTrails apart from previous visualization and scientific workflow systems is a novel action-based mechanism that uniformly captures provenance for data products and workflows used to generate these products. This mechanism not only ensures reproducibility of results, but it also simplifies data exploration by allowing scientists to easily navigate through the space of workflows and parameter settings for an exploration task.

1 Introduction

Workflow systems have been traditionally used to automate repetitive tasks and to ensure reproducibility of results [1,6,9,10]. However, for applications that are exploratory in nature, and in which large parameter spaces need to be investigated, a large number of related workflows must be created. Data exploration and visualization, for example, require scientists to assemble complex workflows that consist of dataset selection, and specification of series of algorithms and visualization techniques to transform, analyze and visualize the data. The workflow specification is then adjusted in an iterative process, as the scientist generates, explores and evaluate hypotheses about the data under study. Often, insight comes from comparing multiple data products. For example, by applying a given visualization process to multiple datasets; by varying the values of simulation parameters; or by applying different variations of a process (e.g., which use different visualization algorithms) to a dataset. This places the burden on the scientist to first generate a data product and then to remember the input data sets, parameter values, and the exact workflow configuration that led to that data product. As a result, much time is spent manually managing these rapidly-evolving workflows, their relationships and associated data.

Consider the problem of radiation treatment planning. Whereas a scanner can create a new dataset in minutes, using advanced dataflow-based visualization tools such as SCIRun [10], it takes from several hours to days to create appropriate visualizations. Fig. 1 shows a series of visualizations generated from a CT scan of a torso—each visualization is created by a different dataflow. During the exploratory process, a visualization expert needs to manually record information about how the dataflows evolve. Often, this is achieved through a combination of written notes and file-naming conventions. For planning the treatment of a

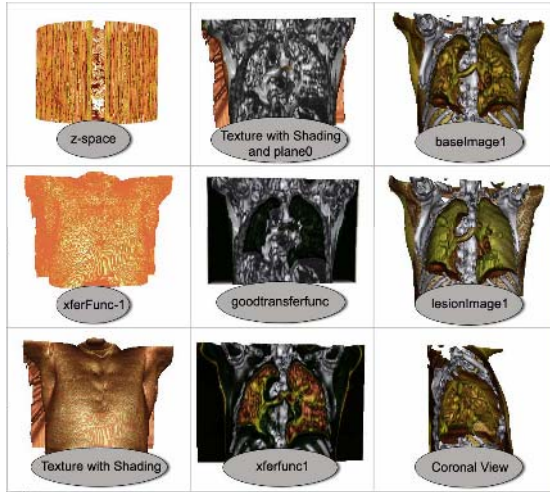


Fig. 1. Series of images generated from an CT scan for planning the radiation treatment of a lung-cancer patient

single patient, it is not uncommon that a few hundred files are created to store dataflow instances and their associated images [2]. To help the radiation oncologists understand the resulting images and ascertain their accuracy, a detailed log of the exact process used to create the images is necessary—this often requires many pages of notes detailing the process.

At the University of Utah, we have been developing VisTrails, a system whose goal is to simplify and streamline the process of scientific data exploration. VisTrails provides an infrastructure which can be combined with and enhance existing visualization and workflow systems. A novel feature of VisTrails is an action-based mechanism which uniformly captures provenance information for both data products and workflows used to generate these products. As shown in Fig. 2, the action-based provenance is stored as a rooted tree, where each node corresponds to a *version* of a workflow, and edges between nodes correspond to the action applied to create one from the other. This tree reflects the process followed by the visualization expert to construct the necessary images, and concisely represents all the workflow versions explored. Although the issue of provenance in the context of scientific workflows has received substantial attention recently, most works focus on data provenance, i.e., maintaining information of how a given data product is generated [6,7,11]. To the best of our knowledge, VisTrails is the first system to provide support for the systematic tracking of workflow evolution.

By maintaining detailed provenance of the exploration process, VisTrails not only ensures reproducibility, but it also allows scientists to easily navigate through the space of workflows and parameter settings used in a given exploration task. In particular, this gives them the ability to return to previous versions

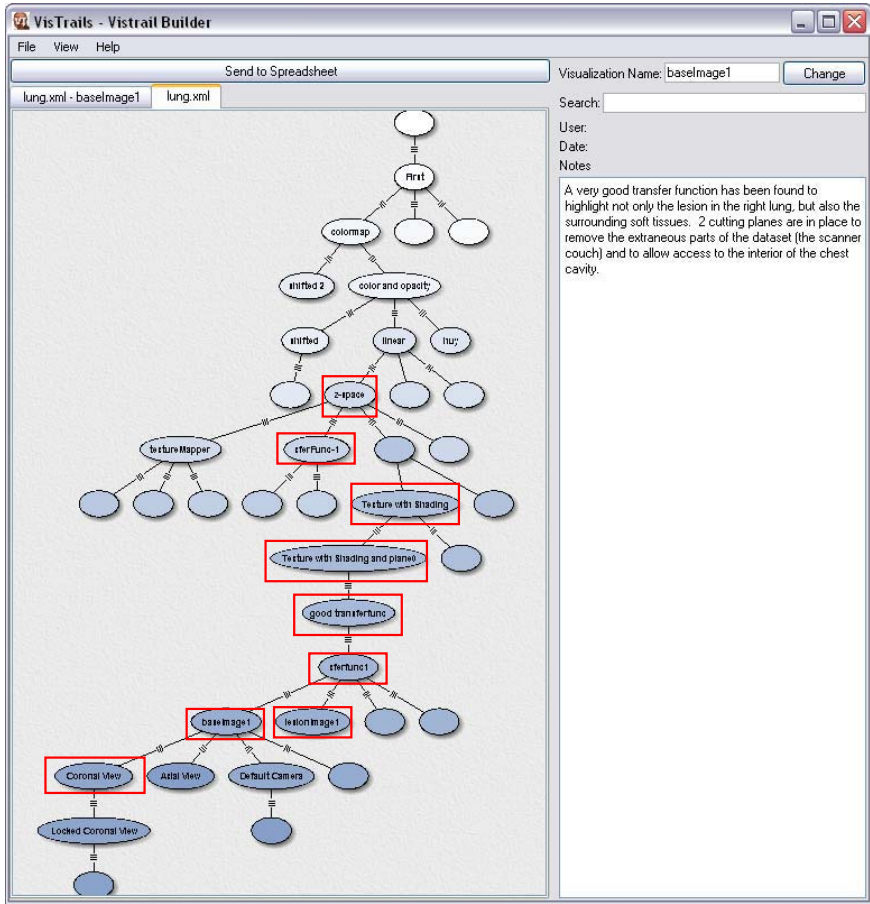


Fig. 2. A snapshot of the VisTrails provenance management interface. Each node in this vistrail version tree represents a workflow version. The nodes highlighted in the tree correspond to the images shown in Fig. 1. This tree captures all the steps followed by a visualization expert to derive the images needed for the radiation treatment planning of a patient.

of a workflow and compare their results. Powerful operations are also possible through direct manipulation of the version tree. These operations, combined with an intuitive interface for comparing the results of different workflows, greatly simplify the scientific discovery process. These include the ability to re-use workflows and workflow fragments through a macro feature; to explore a multi-dimensional slice of the parameter space of a workflow and generate a large number of data products through bulk-updates; to analyze (and visualize) the differences between two workflows; and to support collaborative data exploration in a distributed and disconnected fashion.

Outline. In this paper, we give an overview of VisTrails. The architecture of the system is described in Section 2. In Section 3, we present the action-based provenance mechanism and discuss some of the data exploration operations it enables. We review the related work and conclude in Section 4, where we also outline directions for future work.

2 VisTrails: System Overview

With VisTrails, we aim to give scientists a dramatically improved and simplified process to analyze and visualize large ensembles of simulations and observed phenomena. Although the initial motivation for developing VisTrails was to provide support for data exploration through visualization, the system is extensible and provides infrastructure for managing metadata and processes involved in the creation of data products in general, not just visualizations. The high-level architecture of the system is shown in Fig. 2. Below we briefly describe its key components. For more details, the reader is referred to [3,5].

Users create and edit workflows using the *Vistrail Builder*, which provides a visual programming interface similar to those of visualization and workflow systems [9,10]. The workflow specifications are saved in the *Vistrail Repository*. Users may interact with saved workflows by invoking them through the *Vistrail Server* (e.g., through a Web-based interface) or by importing them into the *Visualization Spreadsheet*. Each cell in the spreadsheet represents a view that corresponds to a workflow instance; users can modify the parameters of a workflow as well as synchronize parameters across different cells. The spreadsheet layout makes efficient use of screen space, and the row/column groupings can conceptually help the user explore the workflow parameter space.

Workflow execution is controlled by the *Vistrail Cache Manager*, which keeps track of invoked operations and their respective parameters. Only new combinations of operations and parameters are requested from the *Vistrail Player*, which executes the operations by invoking the appropriate functions from the *Visualization and Script APIs*. The Player also interacts with the *Optimizer* module, which analyzes and optimizes the workflow specifications.

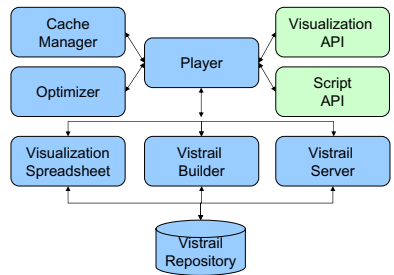


Fig. 3. VisTrails Architecture

3 Action-Based Provenance and Data Exploration

Vistrail: An Evolving Workflow. To provide full provenance of the exploration process, we introduce the notion of a vistrail. *A vistrail captures the evolution of a workflow—all the trial-and-error steps followed to construct a set of data products.* A vistrail consists of a collection of workflows—several versions

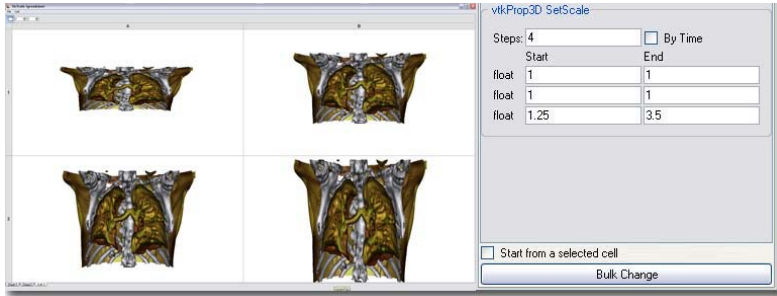


Fig. 4. Results of a bulk update exploration voxel scaling in a single dimension shown in the VisTrails Spreadsheet

of a workflow and its instances. It allows scientists to explore data products by returning to and modifying previous versions of a workflow.

A vistrail is depicted in Fig. 2. Instead of storing a set of related workflows, we store the operations (actions) that are applied to the workflows. A vistrail is essentially a tree in which each node corresponds to a *version* of a workflow, and the edge between nodes P and C, where P is the parent of C, corresponds to one or more actions applied to P to obtain C. More formally, let WF be the domain of all possible workflow instances, where $\emptyset \in WF$ is a special empty workflow. Also, let $x : WF \rightarrow WF$ be a function that transforms a workflow instance into another, and \mathcal{W} be the set of all such functions. A vistrail node corresponds to a workflow f constructed by a sequence of actions x_i , where each $x_i \in \mathcal{W}$:

$$f = x_n \circ x_{n-1} \circ \dots \circ x_1 \circ \emptyset$$

Workflow Change Actions. In the current VisTrails prototype, we implemented a set of operators that correspond to common actions applied to workflows in the exploratory process, including: adding or replacing a module, deleting a module, adding a connection between modules, and setting parameter values. We also have an import operator that adds a workflow to an empty vistrail—this is useful for starting a new exploration process. Internally, the vistrail tree is represented in XML. This allows users to query the workflows and the provenance information, as well as share information easily. For a description of the vistrail schema, see [5].

The action-oriented provenance mechanism captures important information about the exploration process through the very simple process of tracking (and recording) the steps followed by a user. Although quite simple and intuitive, this mechanism has important benefits. Notably, it uniformly captures both changes to workflow instances (i.e., parameter value changes) and to workflow specifications (i.e., changes to modules and connections). In addition, it enables several operations that greatly simplify the data exploration process. We outline some of these operations below, for more details, see [5].

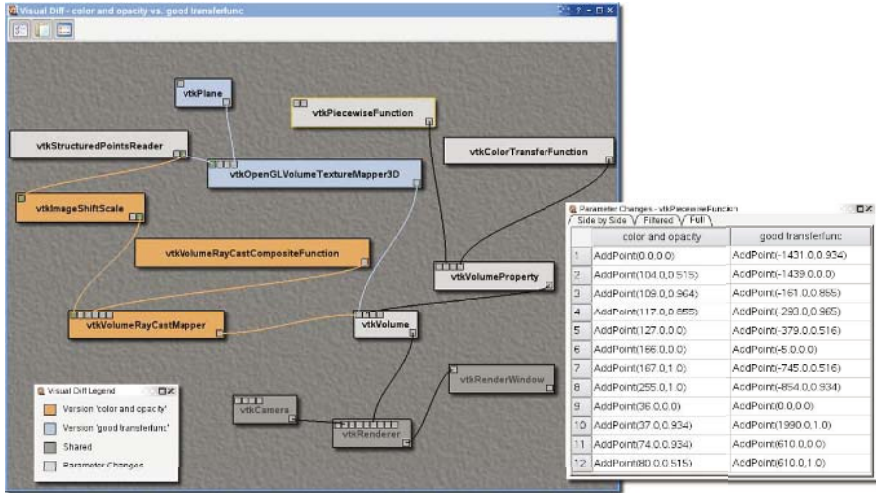


Fig. 5. Visual diff interface. This figure shows the differences between the nodes (workflows) labeled *color and opacity* and *good transferfunc*.

Scalable Derivation of Data Products. The action-oriented model leads to a very natural means to *script* workflows. For example, to execute a given workflow f over a set of n different parameter values, one just needs to apply a sequence of set parameter actions to f :

$$(\text{setParameter}(id_n, \text{value}_n) \circ \dots \circ (\text{setParameter}(id_1, \text{value}_1) \circ f) \dots)$$

Or to compare the results of different data transformation algorithms represented by modules R_1 and R_2 , a *bulk update* can be applied that replaces all occurrences of R_1 with R_2 modules. Fig. 3 shows the VisTrails bulk-change interface. For the workflow corresponding to the node labeled *baseImage1* in the tree of Fig. 2, the user instructs the system to create visualizations varying the voxel size from 1.25 to 3.5 in four steps. VisTrails executes the workflow using the interpolated values and automatically displays the four images in the spreadsheet, where the specialist can easily select the most accurate one. Since some scanners use different resolution in different axes, correcting non-uniform resolution is a common task while dealing with CT scans. To perform this task using SCIRun [10], the visualization expert must go through the lengthy process of manually setting these parameters, one by one through a GUI and saving the resulting images into files.

Re-Use of Stored Provenance. To construct complex scientific workflows, users must have deep knowledge of underlying tools and libraries. Even for experts, creating these workflows can be time-consuming. Thus, mechanisms that allow the re-use of workflows or workflow fragments are key to streamlining the exploratory process. In VisTrails, users can create macros by selecting a sequence of actions in the version tree, or by selecting a workflow fragment. Internally, a

macro m is represented as a sequence of operations $x_j \circ x_{j-1} \circ \dots \circ x_k$. To apply m to a workflow f in the version tree, VisTrails simply composes m with the actions of f .

Interacting with Provenance Information. At any point in time, the scientist can choose to view the entire history of changes, or only the workflows important enough to be given a name, i.e., the tagged nodes in the version tree. The version tree in Fig. 2 represents the history of the changes applied to a workflow to generate the visualizations shown in Fig. 1. Note that in this figure, only tagged nodes are displayed. Edges that hide untagged nodes are marked with three short perpendicular lines. In addition, since the tree structure only shows the dependencies among the workflow versions, different saturation levels are used to indicate the chronological order in which the versions were created—darker nodes are more recent.

To better understand the exploratory process, users often need to compare different workflows. The difference between two nodes in the vistrail tree can be derived by computing the difference between the sequences of actions associated with the nodes. The visual diff interface of VisTrails is illustrated in Fig. 5.

Collaborative Data Exploration. Data exploration is a complex process that requires close collaboration among domain scientists, computer scientists and visualization experts. The ability to collaboratively explore data is key to the scientific discovery process. A distinctive feature of the VisTrails provenance mechanism is monotonicity: nodes in the vistrail version tree are never deleted or modified—once pipeline versions are created, they never change. Having monotonicity makes it possible to adopt a collaboration infrastructure similar to modern version control systems (e.g., GNU Arch, BitKeeper, DARCS). A user’s local copy can act as a repository for other users. This enables scientists to work offline, and only commit changes they perceive as relevant. Scientists can also exchange patches and synchronize their vistrails. The vistrail synchronization algorithm is described in [5].

4 Related Work and Discussion

In this paper, we gave an overview of VisTrails, a system that provides a novel infrastructure for tracking provenance of both data products and workflow evolution. VisTrails is not intended to replace visualization and scientific workflow systems, instead it can be combined with and enhance these systems.

Although provenance in the context of scientific workflows has received substantial attention recently, most works focus on data provenance. To the best of our knowledge, VisTrails is the first system to provide support for tracking workflow evolution. Provenance has also been investigated in other areas. In their pioneering work on the GRASPARC system, Broadlie *et al.* [4] proposed the use of a history mechanism that allowed scientists to steer an ongoing simulation by backtracking a few steps, changing parameters, and resuming execution.

However, their focus was on steering time-dependent simulations, not on data exploration. Kreuzeler *et al.* [8] proposed a history mechanism for exploratory data mining. They used a tree-structure, similar to a vistrail, to represent the change history, and described how undo and redo operations could be calculated in this tree structure. Whereas their theoretical framework attempted to capture the complete state of a software system, VisTrails uses a simpler model and only tracks the evolution of workflows. This allows for the much simpler action-based provenance mechanism described above.

Maintaining detailed provenance has many benefits, but it also presents many challenges. A potential problem is information overflow—too much data can actually confuse users. An important challenge we need to address is how to design intuitive interfaces and provide adequate functionality to help the user interact with and use the provenance information productively. We are currently investigating interfaces and languages that facilitate the querying and exploration of the provenance data as well as efficient storage strategies.

A big barrier to a more wide-spread use of scientific workflow systems has been complexity. Although most systems provide visual programming interfaces, assembling workflows requires deep knowledge of the underlying tools and libraries. This often makes it hard for domain scientists to create workflows and steer the data exploration process. An important goal of our research is to eliminate, or at least reduce this barrier. VisTrails already presents a significant step towards this goal. The existing facilities for scalable parameter exploration and workflow re-use give domain scientists a high degree of flexibility to steer their own investigations. Since VisTrails records all user interactions, an interesting direction we intend to pursue is to try to identify exploration patterns in the version tree and use this knowledge to help users create new workflows and/or solve similar problems.

Acknowledgments. We thank Dr. George Chen (MGH/Harvard University) for providing us the lung datasets, and Erik Anderson for creating the lung visualizations. This work is partially supported by the NSF (under grants IIS-0513692, CCF-0401498, EIA-0323604, CNS-0514485, IIS-0534628, CNS-0528201, OISE-0405402), the DOE, and an IBM Faculty Award. E. Santos is partially supported by a CAPES/Fulbright fellowship.

References

1. G. Alonso and C. Mohan. Workflow management: The next generation of distributed processing tools. In S. Jajodia and L. Kerschberg, editors, *Advanced Transaction Models and Architectures*, chapter 2. Kluwer, 1997.
2. E. Anderson, S. Callahan, G. Chen, J. Freire, E. Santos, C. Scheidegger, C. Silva, and H. Vo. Visualization in radiation oncology: Towards replacing the laboratory notebook. Technical Report UUSCI-2006-017, SCI Institute–Univ. of Utah, 2006.
3. L. Bavoil, S. Callahan, P. Crossno, J. Freire, C. Scheidegger, C. Silva, and H. Vo. VisTrails: Enabling Interactive Multiple-View Visualizations. In *IEEE Visualization 2005*, pages 135–142, 2005.

4. K. Brodlie, A. Poon, H. Wright, L. Brankin, G. Banecki, and A. Gay. GRASPARC: a problem solving environment integrating computation and visualization. In *IEEE Visualization '93*, pages 102–109, 1993.
5. S. Callahan, J. Freire, E. Santos, C. Scheidegger, C. Silva, and H. Vo. Using provenance to streamline data exploration through visualization. Technical Report UUSCI-2006-016, SCI Institute–Univ. of Utah, 2006.
6. I. Foster, J. Voeckler, M. Wilde, and Y. Zhao. Chimera: A virtual data system for representing, querying and automating data derivation. In *Statistical and Scientific Database Management (SSDBM)*, pages 37–46, 2002.
7. P. Groth, S. Miles, W. Fang, S. C. Wong, K.-P. Zauner, and L. Moreau. Recording and using provenance in a protein compressibility experiment. In *Proceedings of the 14th IEEE International Symposium on High Performance Distributed Computing (HPDC'05)*, July 2005.
8. M. Kreuzeler, T. Nocke, and H. Schumann. A history mechanism for visual data mining. In *IEEE Symposium on Information Visualization*, pages 49–56, 2004.
9. B. Ludäscher, I. Altintas, C. Berkley, D. Higgins, E. Jaeger-Frank, M. Jones, E. Lee, J. Tao, and Y. Zhao. Scientific Workflow Management and the Kepler System. *Concurrency and Computation: Practice & Experience*, 2005.
10. S. G. Parker and C. R. Johnson. SCIRun: a scientific programming environment for computational steering. In *Supercomputing*, 1995.
11. Y. L. Simmhan, B. Plale, and D. Gannon. A survey of data provenance in e-science. *SIGMOD Record*, 34(3):31–36, 2005.

Virtual Logbooks and Collaboration in Science and Software Development

Dimitri Bourilkov*, Vaibhav Khandelwal, Archis Kulkarni, and Sanket Totala

University of Florida, Gainesville, FL 32611, USA
bourilkov@phys.ufl.edu

Abstract. A key feature of collaboration is having a *log* of what and how is being done - for private use/reuse and for sharing selected parts with collaborators in today's complex, large scale scientific/software environments. Even better if this log is *automatic*, created on the fly while a scientist or software developer is working in a habitual way, without the need for extra efforts. The CAVES (Collaborative Analysis Versioning Environment System) and CODESH (Collaborative DEvelopment SHell) projects address this problem in a novel way, building on the concepts of *virtual state* and *virtual transition* to provide an automatic persistent logbook for sessions of data analysis or software development in a collaborating group. Repositories of sessions can be managed dynamically to record and make available in a controlled way the knowledge accumulated in the course of a scientific or software endeavor.

Heraclitus asked: How can you bathe in the same river twice?

Quine answers: It's easy, though it is hard to bathe in the same water twice.

1 Introduction

The scientific and software development processes demand the precise tracking of how a project is evolving over time, in order to be able to explore many different alleys simultaneously, moving forward to well defined states when successful - or rolling back otherwise. In this context a virtual session is the process of moving from a well defined initial to a well defined final state. The concept of “virtuality” with respect to existence means that we can define states that may be produced in the future, as well as record the “history” of states that exist now or have existed at some point in the past. By keeping an *automatic log* of initial states and transformations (knowledge about how to transform to desired final states) we have a general tool to track the evolution of a project. Such a tool will be equally useful for software development or any scientific work done on computers. A good logging system will enable a collaborating group to create and/or recreate virtual states on demand. The ability to reproduce a state can have many implications: it may be more practical (e.g. much less space consuming) to keep the initial states and the knowledge than all final states.

* Corresponding Author.

The decomposition in sessions can describe complex processes and procedures as a sequence of many small steps at the desired level of “atomicity”.

The idea of virtual logbooks of sessions complements the idea of virtual data [1,2], where data comes complete with a recipe how to (re)produce it, by putting more emphasis on the interactive aspect of work done by users in their habitual ways, creating the log for the session *automatically, on the fly*, while the work is progressing. There is no need per se to provide any code in advance, but the user can execute/modify preexisting programs if desired. When a piece of work is worth recording, the user logs it in a persistent session repository with a unique identifier for later use/reuse.

Tools like this are vital to facilitate efficient collaboration in today’s large, geographically distributed teams with their needs to be able to advance a project anytime and anywhere without space or time restrictions. Consider e.g. the scenario where thousands of researchers spread over different continents are working together on projects like the Large Hadron Collider [3], the most powerful particle accelerator built so far, expected to start data taking in 2007. In such a scenario, there is a need for efficient means of storing the data and methods used to create this data, and sharing these stored sessions between the collaborators. The CAVES and CODESH projects [4,5,6,7] build tools to address this problem.

2 Project Outline

The basic idea behind the two projects ¹ is the same and they share the same architecture. CAVES is designed specifically for users performing data analysis with the widely popular analysis package ROOT [8]. CODESH is a generalization of the same approach for any type of work done on the command line, like scripting in typical shells, e.g. `bash` or `tcsh`.

The primary use case is a ‘virtual session’. Each user works on a per session basis in an open environment. The work of the user for a session is recorded in the ‘virtual logbook’, while the environment is considered as available or provided by the collaborating group. The splitting between logbook and environment parts is to some extent arbitrary. We will call it a *collaborating contract*, in the sense that it defines the responsibilities of the parties involved in a project. The fundamental concept is to store information in enough detail to provide a share/replay mechanism, optionally modifying the inputs, for user’s sessions at any other place or time by other users.

This is achieved by maintaining a virtual logbook, which records the initial state (pre-conditions of a session), all the commands typed by the user, all the outputs generated and also all the programs executed to produce the results. Also the changes made to the environment i.e. environment variables and aliases are recorded.

When a user’s session ends, or when the user would like to checkpoint the work done so far, he/she tags the complete log, which automatically collects the source program files, and optionally the data used, with a uniquely generated tag and

¹ For more details about the evolution of our design we refer the reader to [4,5].

logs it to a repository. Thus the repositories can contain hundreds and thousands of such stored sessions. Reproduction of a session is possible by extracting the log, data and source files and executing the commands listed in the log files and running the scripts that have been downloaded. Also the environment changes can be carried across sessions.

The repositories of such sessions can be on the local machine for personal usage and also on shared servers for the use of collaborating groups. Generally the user will store all his sessions locally and 'publish' selected important sessions to shared repositories. He/she may also extract and re-produce sessions stored by other collaborators.

There is also a feature, aptly called 'Snapshot', which allows logging entire directory structures under the current working directory. These can later be retrieved and thus provide a virtual working directory. Using this concept, a virtual session can be copied to any place on the same machine or even across machines and re-started or modified. Of course this is possible if the user works relative to the root of the snapshot directory and avoids using absolute paths.

3 Architecture

We have identified three distinct Tiers in the architecture:

- A. The User Tier
- B. The Main CODESH or CAVES Tier
- C. The Backend

Each Tier is completely independent of the other Tiers and it makes use of only the interfaces provided by the other Tiers. Thus we can change one Tier without affecting the operation of the others. The CODESH architecture is shown in Fig. 1. As most details when describing the architecture are common for the two projects, we will concentrate on the CODESH description, mentioning CAVES only where necessary to highlight the distinct nature of each project. Lets examine each Tier in turn:

A. The User Tier: This mainly implements the User Interface. We provide an interface similar to the Unix/Linux command line shell or to the ROOT command line. The user can start his session either in the batch mode or in interactive mode. In the interactive mode, the user types shell (or ROOT) commands just as he/she would on a Unix/Linux shell. He/she also types CODESH (CAVES) commands for the session logging and similar tasks. All the input from the user is parsed and fed to the second Tier, which is the main CODESH (or CAVES) Tier. Also the results produced are displayed on the screen for the user to view.

B. The Main CODESH (CAVES) Tier: This is the heart of our system. It is solely responsible for getting the user input, logging the user sessions, maintaining state information, delegating the shell (ROOT) commands to the under-lying shell (analysis package) and all the communication with the backend Repositories.

Based on the logical separation of the tasks, we have identified 4 different modules that comprise this Tier. They are:

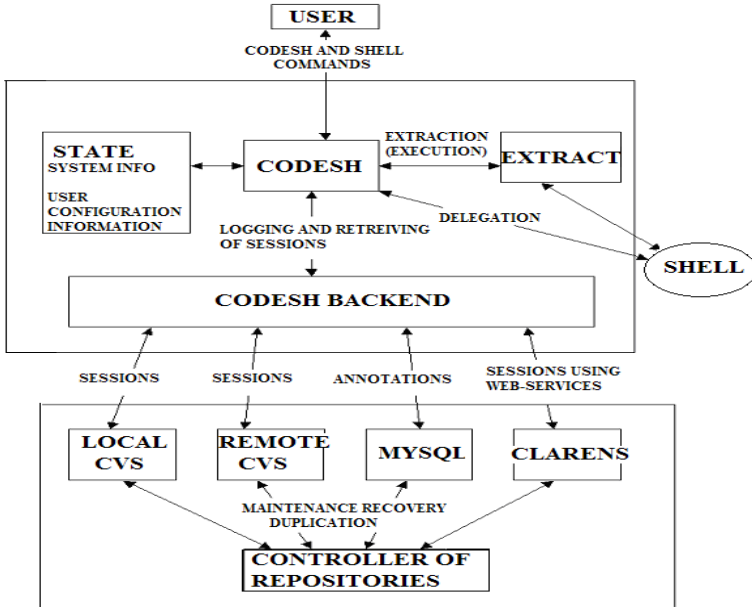


Fig. 1. The scalable and distributed CODESH architecture

i. **CODESH (or CAVES client class)**: It is the main controller module that interacts with all the remaining 3 modules for the successful execution of tasks in this Tier. It delegates shell commands directly to the shell or through the Extract module, which is described later. It also reads and updates the state information stored in the State Information module. It also interacts with the CODESH backend module for the storing and retrieving of the sessions.

ii. **CODESH (CAVES) Backend**: This module interacts with the backend repositories to provide the storage and retrieval of the session information and also to get some status information e.g. a listing of all the sessions that have been stored. It provides a backend independent interface, which is used by the CODESH module.

iii. **State information**: This module stores and maintains all the configuration information during any active user sessions. We broadly classify this state information in two categories:

1. **System information**: This includes the aliases and environment variables that need to be kept track of during the session. We track the changes made to these during the session and provide routines for propagating them and also for logging them along with the session.

2. **User Configuration information**: This includes the various user-selected configurations. Some options provided for customized behavior of CODESH are:

Loglevel: Specifies how much to log

Codeshloglevel: Specifies whether to log CODESH commands in addition to the shell commands which are always logged

Debuglevel: Specifies how much debugging information to print on the screen

Batchmode: Option to enable/disable the batch mode operation

Username: Allows changing the user name used for tagging the sessions

ExtensionList: Maintains a list of all extensions treated as scripts.

iv. Extract module: This module is responsible for the extraction of the sessions i.e. re-executing them and getting the desired outputs. It delegates the shell (ROOT) commands and scripts to the under-lying shell (analysis package) for execution. Currently we support the `bash` and `tcsh` shells. But support for other shells can be easily added.

C. The Backend: The Backend stores the sessions, in such a way that they can be re-created later by some other user who extracts a session. For each session, we store the log files, the source files and optionally the data files. Each session is identified by a unique identifier which consists of 3 parts: the user's name, the current date and time and a user supplied name for the session. We provide support for three different types of backends:

i. CVS: We use CVS [9] as the main backend for storing the sessions. Using the CVS checkin, checkout and other commands we implement our commands like Log session and Extract session (for a comprehensive listing of CODESH commands refer to the next Section). We also provide an option of using Remote CVS as a backend i.e. using a pserver.

ii. Clarens: We provide support for using Clarens [10] as a backend. The Clarens Grid-Enabled Web Services Framework is an open source, secure, high-performance "portal" for ubiquitous access to data and computational resources provided by computing grids. This is a Service Oriented backend that provides services for Authentication, File and Data Transfer etc.

iii. MySQL: We provide for a MySQL backend, which stores only the metadata information regarding each session in the Database. These annotations help in fast searches through stored sessions for some particular session types. After such a session is found, a local or remote CVS repository can be contacted to fetch the complete session.

Every user may have local CVS repositories where he/she stores all personal sessions. Typically the user will want to commit some of the sessions to shared remote repositories and also extract some sessions stored by other users at the shared repositories. Thus we provide support for copying and moving sessions between repositories and also deleting sessions stored at some particular repository. We also plan to provide a way of cloning entire repositories.

Controller of the Repositories: This module takes care of the maintenance, recovery and similar tasks related to the different repositories. Our design structure is distributed in nature and thus we can have numerous controllers instead of just one centralized controller.

4 Typical Usage Scenario

Currently CODESH is implemented in the Python, and CAVES in the C++ programming language. For CAVES the user compiles an executable using the CAVES

code and the `ROOT` libraries. Once started, this executable has all the functionality and behavior of the normal `ROOT` executable, including in addition the `CAVES` commands. Typically a user starts `CODESH` by running the `Codesh.py` file. He can specify the different loglevels and other such customizations in a `Codesh.conf` file or specify them after he runs `CODESH`. In the interactive mode, he then views a command line interface on which he can start his session. Optionally he can use the batch mode and specify a file, which contains all the commands that are to be executed in a batch. To assist in logging his work and re-creating the results of previously stored sessions, he can use the various `CODESH` commands provided. Some of these are:

- i. `Browse`: To list all the stored sessions and also optionally restrict the search depending on date/time or user. Also used to browse metadata associated with the sessions.
- ii. `Inspect`: To view the contents of a particular session and optionally download all the source files.
- iii. `Extract`: To execute a stored session and re-create the results.
- iv. `Log`: To log a session between user defined checkpoints along with (optionally) all or some of the programs executed during the session. The level of logging depends on user defined log levels for that particular session.
- v. `Tagcopy`: To copy a stored session from the source repository to the destination repository.
- vi. `Tagdelete`: To delete a stored session.
- vii. `Takesnapshot`: To work in a separate sandbox and store the entire sandbox in a repository. This complete sandbox i.e. all the files and directories under the working directory, can then be retrieved later by the same user or some other user and worked upon.
- viii. `Getsnapshot`: To retrieve any previously stored sandbox. This is very useful in cases where a previously stored sandbox can be re-created at various places, by various people and all of them can start working from the place where the original user had left.
- ix. `Browsesnapshots`: This command lists all (or a selected subset of) the sandboxes committed by all the users.
- x. `Setenv`, `Getenv`: To access and modify environment variables independent of the underlying shell.
- xi. `GetAlias`: To get all the active aliases.

5 Test Results

Tools like `CODESH` or `CAVES`, designed to be used for collaborative development by many users, have to deal with different styles or work preferences and customizations by many individual users. These differences can be e.g. in the underlying shells used, level of logging/debugging desired, kind of work, user permissions and so on. We have developed `CODESH` and `CAVES` as flexible tools, easy to customize and extend with new user defined commands. We have tried to exhaustively test them with many different customizations using as backend local or remote `CVS` repositories.

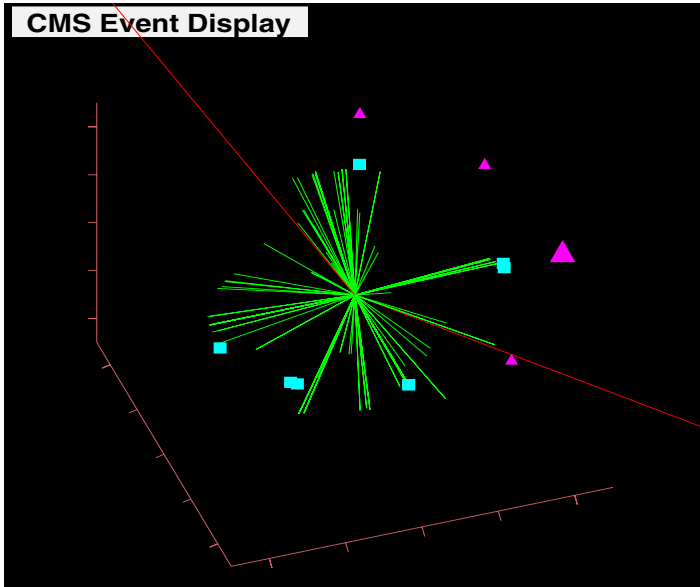


Fig. 2. Event display from a Monte Carlo simulation for the CMS experiment. Charged particle tracks are shown in green, muon tracks in red, reconstructed jets with different algorithms as triangles and squares.

We have also done Stress testing for scenarios where the size and quantity of the logged data was really overwhelming. We have tested CODESH till the size of the repository was 10,000 sessions. Our code is resilient enough that even with 10,000 sessions in the repository the performance was only marginally slower than with very few sessions in the repository. All the sessions stored were of similar type and size. Specifically with 10 sessions stored in the repository, the inspection of a session took around 1 second, and even with 10,000 sessions stored, the inspection of a session took only 2 seconds on a 1 GHz Pentium III machine for a local repository.

We have built a fully distributed data analysis system based on ROOT and CAVES. The virtual sessions are stored in local or remote (pserver based) CVS repositories. The input and output datasets are stored using xrootd [11] servers. In this way users can browse, inspect and reproduce the sessions of their colleagues starting from a completely clean slate on a new desk- or laptop. All the necessary knowledge, code and data are delivered from the remote servers. An example of an event display from a Monte Carlo simulation for the CMS experiment [12] at LHC produced in this way is shown in Fig. 2.

6 Related Work

Archives typically keep final states. Often it is unclear how they were created. In computers the initial and final states are transient. Knowledge is not recorded

systematically. Paper logs are hard to produce, hard to search, hard to share. They need transfer to a computer before making the knowledge widely available.

The history mechanism in typical shells like `tcsh` or `bash` logs a pre-defined number of commands in a file. But it provides no persistency mechanism for storing sessions or for exchanging them between collaborators. The scripts executed during a session, the pre- and post- conditions are not logged. The script [13] utility goes one step further, logging the standard output from the commands as well, all the rest is left to the user. Our automatic logbook/virtual sessions approach does much more by managing private and shared repositories of complete session logs, including the commands and the programs, and the ability to reproduce the results or reuse them for future developments.

In [14], contrary to most existing provenance systems which use disclosed provenance like annotations, transformations or workflows, an observed provenance approach at the kernel and system call level is developed. The authors note the desirability of combining the full semantic knowledge of disclosed provenance and the automatic and transparent collection of observed provenance systems, noting also the substantial challenges like provenance granularity, versioning, provenance pruning, overheads etc. Our approach taken with the CODESH project offers flexibility and good balance between observed and disclosed provenance: the users of our system can select the splitting between logbook and environment parts, which we call *collaborating contract*, depending on their needs. We provide observed provenance at the shell level, which, due to its proximity to the users, offers a rich semantic knowledge by seamlessly observing the work done in a virtual session. The adoption of a versioning system like CVS as a persistent backend helps in solving the versioning problem and provides an elegant approach to pruning by only storing the differences between a potentially large number of similar sessions. In summary, the CODESH project combines in a natural way some of the key desired features of the two extremes outlined above.

7 Current and Future Work

We have used CODESH to record the production and analysis of data for large scale simulations in high energy physics, and for various software development and configuration tasks at several locations distributed across the United States. CAVES was used to record analysis sessions of the produced data, including analyses demonstrated at Supercomputing 2005 in Seattle.

Our ongoing and future work consists e.g. of implementing: all the robust functionality available with the CVS backend in the more difficult case of the web service based Clarens backend; the full set of administrative tasks for the management, maintenance and control of private and shared repositories; Web interfaces for users to browse through the repositories; and a utility by which the user can clone entire repositories. Public releases of the first functional systems for automatic logging in a typical working session will be available from [15], and the projects are hosted as open source [16]. In addition we are working on automatically converting session logs to workflows, and the ability to develop locally and seamlessly schedule more CPU/data intensive tasks on grid infrastructure.

In summary, our projects take a pragmatic approach in assessing the needs of a community of scientists or software developers by building series of working prototypes with increasing sophistication. By extending with automatic logbook capabilities the functionality of a typical UNIX shell (like `tcsh` or `bash`) - the CODESH project, or a popular analysis package as ROOT - the CAVES project, these prototypes provide an easy and habitual entry point for researchers to explore new concepts in real life applications and to give valuable feedback for refining the system design.

The study is supported in part by the United States National Science Foundation under grants NSF ITR-0086044 (GriPhyN) and NSF 0427110 (UltraLight).

References

1. I. Foster *et al.*, presented at the 14th International Conference on Scientific and Statistical Database Management (SSDBM 2002), Edinburgh, 2002; GriPhyN Technical Report 2002-7, 2002.
2. I. Foster *et al.*, Proceedings of CIDR 2003 - Conference on Innovative Data Research; GriPhyN Technical Report 2002-18, 2002.
3. The Large Hadron Collider close to Geneva, Switzerland, will collide proton-proton beams at energies of 14 TeV starting in 2007; <http://lhc-new-homepage.web.cern.ch/lhc-new-homepage/>.
4. D. Bourilkov, <http://arxiv.org/abs/physics/0401007>, arXiv:physics/0401007.
5. D. Bourilkov, <http://arxiv.org/abs/physics/0410226>, Int. J. Mod. Phys. A **20** (2005) 3889 [arXiv:physics/0410226].
6. D. Bourilkov, ICCS 2005 conference, Atlanta, USA, 2005; V.S.Sunderam et al. (Eds.): ICCS 2005, LNCS 3516, pp. 342-345, 2005, Springer Verlag Berlin Heidelberg.
7. D. Bourilkov and V. Khandelwal, WMSCI 2005 conference, Orlando, USA, 2005; published in the Proceedings, ed. N.Callaos, W.Lesso and K.Horimoto, ISBN 980-6560-60-4, vol. VIII, p.175, IIS 2005.
8. Brun, R. and Rademakers, F.: ROOT - An Object Oriented Data Analysis Framework. Nucl. Inst. & Meth. in Phys. Res. **A 389** (1997) 81-86
9. CVS: The Concurrent Versions System CVS, <http://www.cvshome.org/>.
10. C. Steenberg et al., Computing in High-Energy and Nuclear Physics (CHEP 03), La Jolla, California, 24-28 Mar 2003; Published in eConf C0303241:MONT008, 2003; e-Print Archive: cs.dc/0306002; <http://clarens.sourceforge.net/>.
11. xrootd home page, <http://xrootd.slac.stanford.edu/>.
12. The CMS experiment at CERN, <http://cms.cern.ch/iCMS/>.
13. The script utility appeared in Berkeley Unix 3.0BSD.
14. Uri Braun et al., "Issues in Automatic Provenance Collection", this proceedings.
15. CODESH/CAVES home page, <http://cern.ch/bourilkov/caves.html>.
16. <https://sourceforge.net/projects/codesh>

Applying Provenance in Distributed Organ Transplant Management

Sergio Álvarez¹, Javier Vázquez-Salceda¹,
Tamás Kifor², László Z. Varga², and Steven Willmott¹

¹ Knowledge Engineering and Machine Learning Group,
Universitat Politècnica de Catalunya. Jordi Girona 1-3, Barcelona, Spain
{salvarez, jvazquez, steve}@lsi.upc.edu
<http://www.lsi.upc.edu/~webia/KEMLG>

² Computer and Automation Research Institute, Kende u. 13-17, 1111 Budapest, Hungary
{tamas.kifor, laszlo.varga}@sztaki.hu
<http://www.sztaki.hu/>

Abstract. The use of ICT solutions applied to Healthcare in distributed scenarios should not only provide improvements in the distributed processes and services they are targeted to assist but also provide ways to trace all the meaningful events and decisions taken in such distributed scenario. *Provenance* is an innovative way to trace such events and decisions in Distributed Health Care Systems, by providing ways to recover the origin of the collected data from the patients and/or the medical processes. Here we present a work in progress to apply *provenance* in the domain of distributed organ transplant management.

1 Introduction

Cooperation among people using electronic information and techniques is more and more common practice in every field including healthcare applications as well. In the case of distributed medical applications the data (containing the healthcare history of a single patient), the workflow (of the corresponding processes carried out to that patient) and the logs (recording meaningful events) are distributed among several heterogeneous and autonomous information systems. These information systems are under the authorities of different healthcare actors like general practitioners, hospitals, hospital departments, etc. which form disconnected *islands of information*. In order to provide better healthcare services, the treatment of the patient typically requires viewing these pieces of workflow and data as a whole.

Also, having an integrated view of the workflow execution and the logs may become important in order to analyse the performance of distributed healthcare services, and to be able to carry out audits of the system to assess if needed, that for a given patient the proper decisions were made and the proper procedures were followed. For all that there is a need to be able to trace back the origins of these decisions and processes, the information that was available at each step, and where all these come from. In order to support this in this paper we propose to make distributed medical applications *provenance-aware*. Our working definition for *provenance* is the following: “the provenance of a piece of data is the process that led to the data” [1,2]. Provenance

2.1 Distributed Transplant Management: The OTM Application

The Organ Transplant Management (OTM) Application aims to speed up the allocation process of solid organs to improve graft survival rates. Its policy implements the Spanish guidelines for organ and tissue procurement and Spanish regulations for allocation, as Spain is world leader in the area, followed as a model by other countries. OTM uses standard web service technology and has been adapted to be provenance-aware, by interacting with the provenance stores in order to keep track of the distributed execution of the allocation process for audit purposes.

Figure 1 summarizes the different administrative domains (solid boxes) and units (dashed boxes) that are modelled in the OTM application. Each of these interact with each other through Web Service interfaces (circles) that send or receive messages. The Organ Transplant Authority (OTA) is an administrative domain with no internal units. In a transplantation management scenario, one or more hospital units may be involved: the hospital transplant unit, one or several units that provide laboratory tests and the unit that is responsible for the patient records (which will use the EHCR application services, see section 2.2). The diagram also shows some of the data stores that are involved: apart of the patient records, these include stores for the transplant units and the OTA recipient waiting lists (WL). Hospitals that are the origin of a donation also keep records of the donations performed, while hospitals that are recipients of the donation may include such information in the recipient's patient record. The OTA has its own records of each donation, stored case by case.

By transforming OTM into a *provenance-aware* application, we augment OTM with a capability to produce at run-time an explicit representation of the process actually taking place (see example in Figure 2). Such representation can be then queried and analysed in order to extract valuable information to validate, e.g., the decisions taken in a given case, or to make an audit of the system over a period of time.

2.2 Medical Record Management: The EHCR System

The Electronic Health Care Record System (EHCR) provides a way to manage electronic health records distributed in different institutions. The architecture provides the structures to build a part of or the entire patient's healthcare record drawn from any number of heterogeneous databases systems in order to exchange it with other healthcare information systems. The EHCR architecture has two external interfaces: a) a Web Service that receives and sends messages (following ENV13606 pre-standard format [3]) for remote medical applications; and b) a Java API for local medical applications that can be used to access the EHCR store directly. The application also uses an authentication Web Service to authorize request messages from remote health care parties.

Making the EHCR system *provenance-aware* provides a way to have a unified view of a patient's medical record with its provenance (i.e. to connect each part of the medical record with the processes in the real world that originated it and/or the individuals, teams or units responsible for each piece of data).

3 Provenance Handling in the OTM Application Domain

The Provenance architecture developed within the PROVENANCE project [1] assumes that the distributed system can be modelled using a service-oriented approach. In this abstract view, interactions with services (seen as *actors*) take place using messages that are constructed in accordance with service interface specifications.

In the case of the OTM application, each organisational unit (the transplant unit, the ER unit, the laboratories) is represented by a service. Staff members of each unit can connect to the unit services by means of GUI interfaces. The provenance of a data item is represented by a set of *p-assertions*, documenting steps of the process, and they are stored and managed in *provenance stores*. The distributed execution of the OTM services is modeled as the interaction between the actors representing the services, and recorded as *interaction p-assertions* (assertions of the contents of a message by the actor that sent or received it) and *relationship p-assertions* (assertions that describe how the actor obtained an interactions' output data by applying some function to input data from other interactions). As in the OTM scenario a decision depends on a human making the decision, additional *actor state p-assertions* (assertions made by actors about their internal state in the context of a specific interaction) are recorded, containing further information on why the particular decision was made and, if available, the identities of the team members involved in the decision.

The application of the provenance architecture to the OTM system had to overcome two challenging issues: a) the provenance of most of the data is *not* a computational service, but decisions and actions carried out by *real* people in the *real* world; b) past treatments of a given patient in other institutions may be relevant to the current decisions in the current institution, so p-assertions about the processes underwent in those previous treatments should be connected somehow to the current p-assertions. An example on how we deal with both issues can be found in section 3.2.

3.1 Provenance Questions

In both the OTM and the EHCR systems, the provenance architecture should be able to answer the following kind of questions, related to a given patient (donor or recipient) or to the fate of a given organ:

- where did medical information used on each step of the process came from,
- which medical actor was the source of information.
- what kind of medical record was available to actors on each step of the process
- when a given medical process was carried out, and who was responsible for it.
- when a decision was taken, and what was the basis of the decision
- which medical actors were asked to provide medical data for a decision
- which medical actor refused to provide medical data for a decision

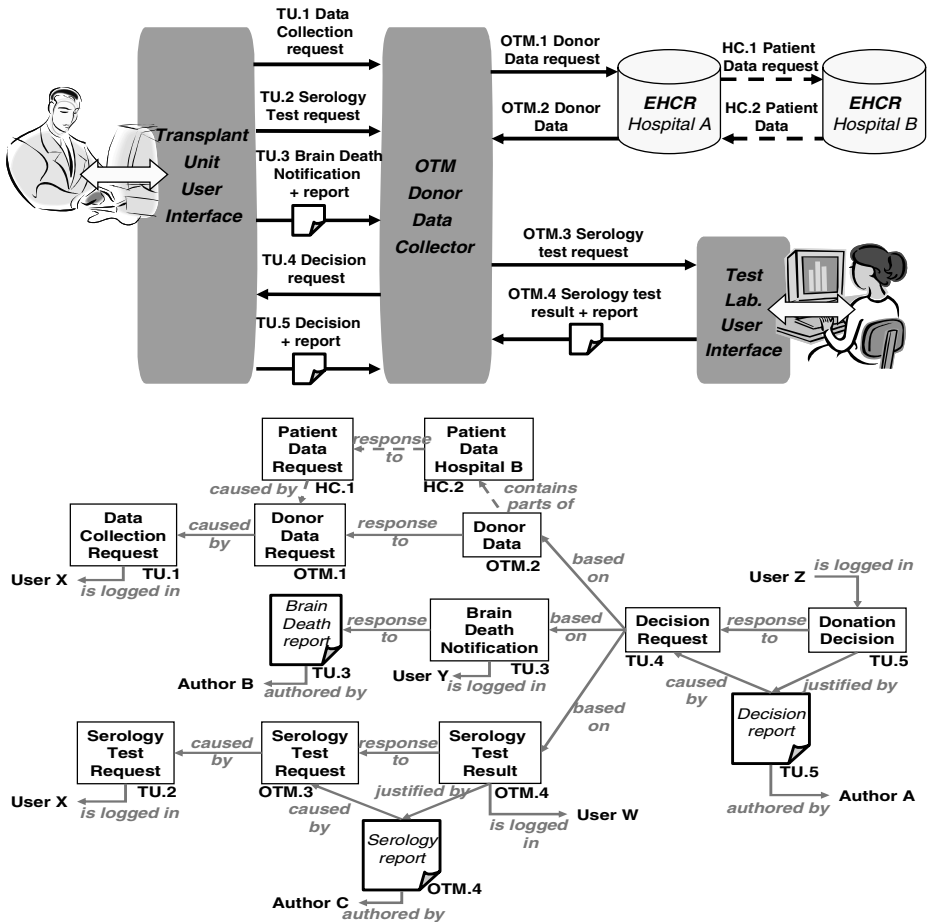


Fig. 2. Example scenario: (top) Interactions of the OTM components involved in a donation decision; (bottom) DAG showing the provenance of the donation decision

3.2 An Example

To illustrate how provenance is handled in the OTM application, let us see how the provenance of a medical decision is recorded and then queried. Figure 2 (top) shows a simplified view over a subset of the donation process. We consider a patient who has previously given consent to donate his organs. As the patient’s health declines and in foresight of a potential organ donation, one of the doctors requests the full health record for the patient and then orders a serology test¹ through the OTM application. After brain death is observed and logged into the system (along with the report certifying the brain death), if all requested data and analysis results have been obtained, a

¹ A serology test is performed over blood samples to detect viruses (HIV, Hepatitis B/C, syphilis, herpes or Epstein-Barr virus) which, if present in the organ, can pass to the recipient.

doctor is asked to make a decision about the patient being a potential donor. This decision is explained in a report that is submitted as justification.

Figure 2 (top) shows the OTM components for this small scenario and their interactions. The Transplant Unit User Interface passes requests (TU.1, TU.2) to the OTM Donor Data Collector service, which gets the electronic record from the EHCR system (OTM.1, OTM.2). Sometimes all or parts of the record are not in the same institution but located in another institution (HC.1, HC.2). The Donor Data Collector service also sends the request for a serology test to the laboratory and gets back the result (OTM.4), along with a detailed report of the test. Reports are also passed in the case of the Brain Death notification (TU.3) and the final decision report (TU.5).

Figure 2 (bottom) graphically represents the subset of the p-assertions produced by the *provenance-aware* OTM which are related to the donation decision. The part of the process that happens within the electronic system is represented by interaction p-assertions (regular boxes) for all interactions (TU.x, OTM.x, HC.x), and relationship p-assertions (*response_to*, *caused_by*, *based_on*) capturing dependencies between data. Even though what happens in the system has a parallelism to what happens in the real world, as we already said this is not enough to fully answer which is the provenance of a given decision. To solve this, we connect the electronic process to the real world by adding actor state p-assertions stating who logged the information in the system (*is_logged_in*) and when (not shown in picture), which are the reports that justify a given state in the system (*justified_by*), who are the authors of these reports (*authored_by*) and when the action reported was performed or the decision taken (not shown). Following back the p-assertions graph in Figure 2 we can trace the provenance of the donation decision, how it was based in some data and test requests, how a brain death notification is also involved, who requested the information, where it came from (in some cases it might come from the EHCR from another hospital), who authored the justifying reports in the main steps of the process.

In those cases (as in Figure 2) where the decision might be based on medical data coming from tests and medical treatments carried out in other institutions, another issue to solve is the following: how to find, retrieve and incorporate the provenance of the data coming from the other institution? If the provenance stores of the different institutions are connected, to solve the aforementioned problem is to solve the issue of discovering the different p-assertions related to the same patient. If this discovery step is done, then actors can make p-assertions that link together the separated sets of p-assertions to create a larger provenance document providing an integrated view of the healthcare history of the patient. The discovery can be done with the help of a patient identifier known to all actors. For privacy reasons the patient identity has to be anonymised. In the OTM application the EHCR system adds case identifiers (identifiers created at run-time) inside the p-assertions to create connections between sets of p-assertions related to the same patient. The result (not shown on Figure 2) would be that the provenance of *Patient Data Hospital B* would be added to the DAG as part of the provenance of the Donation Decision. Linking provenance stores in different administrative domains raises some challenging issues on privacy and security, though (see [4] for more details).

We had to find equilibrium between the amount of collected data and the level of interference such data collection may cause in the real medical process. The use of the reports and the information logged by the staff does not give full information about

what happens in real world, but gives more than enough information to trace the individual or team involved, while not introducing an excessive increase of workload on the medical staff (we use the same reports medical staff already produces). It is important to note that the person who is logged in might not always be who authors the justifying reports (both are recorded in OTM), and the time when things are reported to the system might not be the time when things have happened (both also recorded in OTM). This is common practice in medical teams: most of reporting is delegated to a team member having the proper credentials and time to do it,² although the report may be later checked and even signed by a prominent member of the team.

4 Related Work

In those first investigations which started to record the origin and history of a piece of data, the concept was called *lineage*. In the SDTS standard [5], lineage was a kind of audit trail that traces each step in sourcing, moving, and processing data, mainly related to a single data item, a logical data record, a subset of a database, or to an entire database [6, 7]. There was also relationship to versioning [8] and data warehouses [9]. The provenance concept was later further explored within the GriPhyN project [10]. These techniques were used in [11] in two respects: 1) data was not necessarily stored in databases and the operations used to derive data items might have been arbitrary computations; and 2) issues relating to the automated generation and scheduling of the computations required to instantiate data products were also addressed. The PROVENANCE project [1] builds on these concepts to conceive and implement industrial strength open provenance architecture for grid systems, including tools for managing and querying provenance stores along with high-level reasoning capabilities over provenance traces. The price to pay for this is that applications should be adapted in order to provide high-quality p-assertions that not only record their inputs and outputs but also the (causal, functional) relation between them. The alternative would be the use of automatic data harvesting techniques such as RDF tuples harvesting [12], where RDF tuples include attribution (*who*) and time (*when*) information which is then processed by an external inference engine in order to construct RDF graphs by some kind of extended temporal reasoning. Another alternatives reduce to a minimum the adaptation step needed to make an application provenance-aware by adding a middleware layer or an execution platform capable to automatically create provenance assertions from the captured events and actions [13,14]. Problem is that, in automatic provenance collectors, it is very hard to infer causal relationships by only comparing sources and times [15], sometimes with the extra help of some derivation rules [16] or rigid workflow definitions. In the case of the medical domain this is not enough. Returning to the example on figure 2, let us suppose that at time $t1$ system records a donor data request from user X for patient P , at time $t2$ it records a serology

² Records of the process may be done asynchronously to avoid delays in critical steps: for instance, a surgeon should not stop a surgery to record through the GUI interface his last decisions and actions taken. If there is enough personnel in the surgery room, an assistant will record the events and decisions in parallel; if not, recording is done after the surgery.

test request from user X for patient P , at time t_3 it records a donation decision from user A for patient P , and $t_1 < t_3$, $t_2 < t_3$. Even if users A and X are the same, it would be unwise to directly infer that the donation decision was based on the result of the donor data request and the result of the serology test request just because i) all refer to the same patient P and ii) both requests happened before the decision was made, as this may not be true in all cases (e.g., anything terribly wrong in the serology test would lead directly to a donation rejection without having to take into consideration the rest of the collected donor data). Adding some generic rules to the temporal reasoner to express on which sources a donation decision uses to be made would not solve the problem either, as these rules can hardly handle all exceptional cases. The solution is to include in the provenance representation an explicit way to express relationships between recorded assertions (e.g. the doctor ticks on screen some boxes indicating which parts of the donor data and which test results he based his decision on, and this is automatically translated by the adapted provenance-aware OTM application into several, very precise, *based_on* relationship p-assertions, valid for that specific case).

In organ allocation management, there are few IT solutions giving powerful support to the allocation of human organs. The EUROTRANSPLANT system [17] is a centralised system where all information and decisions are made in a central server, and all activity is recorded in standard logging systems. The *Swisstransplant* system [18], is a distributed system which combines agent technology and constraint satisfaction techniques for decision making support in organ transplant centers. In this case all activity is also recorded in standard logging systems. Up to our knowledge, the application of provenance techniques to distributed transplant management is novel.

5 Conclusions and Ongoing Work

In this paper we present an application of a service-oriented architecture for provenance applied in distributed medical systems. We used as example the domain of human organ allocation for transplantation purposes, where provenance is used to trace the actors that were involved in the important steps of the process (e.g., a medical decision) and to provide an integrated view of the medical history of a patient through the recollection of the medical treatment processes carried out in one or several institutions. In the context of the PROVENANCE project we are building a first demonstrator of this application. Evaluation is planned with some hospital and transplant coordinators in Spain, who will give us feedback in the last steps of the development and fine-tuning of the application.

Acknowledgements

This work has been funded mainly by the IST-2002-511085 PROVENANCE project. Javier Vázquez-Salceda's work has been also partially funded by the "Ramón y Cajal" program of the Spanish Ministry of Education and Science. All the authors would like to thank the PROVENANCE project partners for their inputs to this work.

References

1. The EU Provenance Project Enabling and Supporting Provenance in Grids for Complex Problems (IST 511085), <http://www.gridprovenance.org/>
2. P. Groth, S. Jiang, S. Miles, S. Munroe, V. Tan, S. Tsalakou, L. Moreau, D3.1.1: An Architecture for Provenance Systems. Technical report, University of Southampton, February 2006. <http://eprints.ecs.soton.ac.uk/12023/>
3. CEN/TC251 WG I: Health Informatics-Electronic Healthcare Record Communication-Part 1: Extended architecture and domain model, Final Draft prENV13606-1 (1999).
4. T. Kifor, L.Z. Varga, S. Álvarez, J. Vázquez-Salceda and S. Willmott. "Privacy Issues of Provenance in Electronic Healthcare Record Systems". Proceedings of the 1st Int. Workshop on Privacy and Security in Agent-based Collaborative Environments (PSACE 2006).
5. American National Standard for Information Systems. Spatial Data Transfer Standard (SDTS) - Part 1, Logical Specifications, Secretariat, United States Geological Survey, National Mapping Division, DRAFT for Review, November 20, 1997, http://mcmweb.er.usgs.gov/sdts/SDTS_standard_nov97/part1b12.html
6. Buneman, P., Khanna, S. and Tan, W.-C., "Why and Where: A Characterization of Data Provenance" in International Conference on Database Theory, (2001).
7. A. Woodruff, and M. Stonebraker. "Supporting Fine-Grained Data Lineage in a Database Visualization Environment", Computer Science Division, U. of California Berkeley, 1997.
8. A. Marian, S. Abiteboul, G. Cobena, and L. Mignet. "Change-Centric Management of Versions in an XML Warehouse", in 27th Int. Conf. of Very Large Data Bases, (2001).
9. Y. Cui, J. Widom and J.L. Wiener. "Tracing the Lineage of View Data in a Warehousing Environment", ACM Transactions on Database Systems, 25 (2). 179–227.
10. The GriPhyN Project. <http://www.griphyn.org>
11. I. Foster, J. Vockler, M. Wilde, Y. Zhao. "The virtual data grid: A new model and architecture for data-intensive collaboration. In: Proceedings of the First Biennial Conference on Innovative Data Systems Research, CIDR 2003, Asilomar, CA, January 5-8, 2003
12. J. Futrelle. "Harvesting RDF triples". International Provenance and Annotation Workshop, Chicago, IL, May 2005.
13. R. Barga. "Automatic Generation of Workflow Execution Provenance". International Provenance and Annotation Workshop, Chicago, IL, May 2006.
14. K.-K. Muniswamy-Reddy, D. A. Holland, U. Braun, and M. Seltzer. Provenance aware storage systems. In Proc. of the 2006 USENIX Annual Technical Conference, June 2006.
15. U. Braun, S. Garfinkel, D. A. Holland, K.-K. Muniswamy-Reddy, M. I. Seltzer. "Issues in Automatic Provenance Collection". In: Proceedings of the International Provenance and Annotation Workshop, Chicago, IL, May 2005.
16. I. Foster, J. Voekler, M. Wilde, Y. Zhao: Chimera: A Virtual Data System for Representing, Querying and Automating Data Derivation. In Proceedings of the 14th Conference on Scientific and Statistical Database Management, 2002.
17. Eurotransplant International Foundation. <http://www.eurotransplant.nl/>
18. Swisstransplant. <http://www.swisstransplant.org/>

Provenance Implementation in a Scientific Simulation Environment

Guy K. Kloss and Andreas Schreiber

Simulation and Software Technology
German Aerospace Center
51147 Cologne, Germany
{Guy.Kloss, Andreas.Schreiber}@dlr.de
<http://www.dlr.de/sc/>

Abstract. Many of today's engineering applications for simulations are lacking mechanisms to trace the generation of results and the underlying processes. Especially computations conducted in distributed computing environments as Grids are lacking suitable means to keep track of used resources. Trust of engineers in results produced within distributed simulation environments is very limited without this information.

This paper will demonstrate how trust and confidence in simulation results could be achieved for engineering applications. It will highlight the backgrounds of the application, of provenance recording, the mapping to the application, and finally the implementation of provenance awareness for the application. Additionally it will present examples of analyzing the information stored to be of further use to the engineer.

1 Introduction

Complex numerical simulation plays an important role in today's industrial development. During the design processes it reduces the amount of expensive validation using real world models for physical examination. This again reduces the time for obtaining high quality designs, and thus gives the engineer the opportunity to evaluate more competing designs.

In this engineering process vast amounts of data are generated. Managing this data in the first place is a problem by itself, but it is subject to different solutions. The specific task to be discussed in this paper is to give the engineer tools at hand to keep track of the history of it's generation, which is important to trust the generated data and to analyze data with respect to changes in the simulation process.

1.1 Provenance

If an organization wants to prove compliance, they must establish the origin and authenticity of the information produced by their processes. This type of documentation needs to be recorded as the process takes place. To have

proven integrity, it must be transparent and auditable. A documentation with the mentioned properties reveals the full history of the creation some information: it's *provenance*. The goal is to conceive a *computer based representation* of provenance, that allows us to perform useful analysis and reasoning. For inter-operability with other applications and adaptability to new regulations, the provenance implementation needs to be open and standardized, rather than closed and monolithic.

The implementation of a provenance architecture demands a high flexibility and robustness to different scenarios. In the *Grid Provenance* project [1] two sample applications with complementary requirements regarding performance and scalability are to be implemented. The organ transplant management (OTM) application relies on a wide geographical distribution of the system, extremely high standards for security and privacy, multiple provenance stores, a high degree of human interaction, and long periods of time until a management process can be considered completed. On the other hand the aerospace engineering application presents high requirements in computation performance within secured networks. A complete set of requirements [2] for the reference implementation has been identified.

1.2 Application

Our goal is to add Provenance awareness to an engineering application which is composed of several individual numerical codes which are integrated in an interactive simulation environment.

The application is used by engineers in the simulation of complex flight maneuvers. This simulation implies the unsteady aerodynamics of a free flying, fully configured, elastic combat aircraft. In this application, two reasons for provenance recording are essential:

- Seamless process documentation for compliance and liability reasons.
- Providing better insight into archived simulational data through analysis by means of complex query methods.

In the following we will first give a very brief overview of the application. After that, an introduction to computer based provenance concepts and the provenance software infrastructure being used is given. Then in Sect. 4.1 we will describe the deployment of provenance services to our application.

2 Engineering Application

Aerospace engineering design problems are often being treated by complex simulations using a workflow of a variety of numerical codes and supporting tools. Usually, these simulation workflow does also contain loops, conditional constructs and parallelization of tasks.

For practically perform such complex simulations, all codes are being integrated into an integration system which provides a graphical user interface, starts all codes in the distribution environment and manages all involved data.

2.1 Simulation of a Maneuvering Combat Aircraft

Our example application for simulation of a maneuvering combat aircraft consist of three major numerical codes (see [3] for details):

- The computational fluid dynamics solver TAU [4] for aero-dynamics,
- the structure mechanics solver NASTRAN for aero-elasticity, and
- the flight mechanics solver SIMULA [5].

Each of these codes have different and very specific constraints and requirements on the underlying hardware and software infrastructure, which requires to start each code on a different machine (see Fig. 1).

2.2 Integration System

An integration system provide a comfortable execution environment for complex simulations. Via a GUI on a desktop computer, the engineer graphically constructs the workflow, sets input parameters for individual components, and controls the workflow execution.

We are using the integration system TENT [6,7] which allows one to integrate all necessary computational components and to construct workflows with these components. It allows engineers to steer and monitor running simulations interactively on the Grid and has an integrated data management systems based on XML and WebDAV.

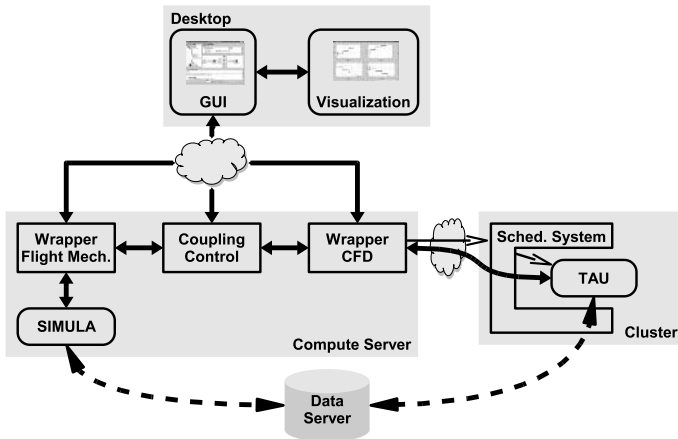


Fig. 1. Distribution of workflow components on the network

3 Implementing Provenance

Sect. 1.1 gives an idea for the requirements for the provenance implementation in engineering tasks. Data history needs to be documented *as events take*

place. This needs to be dealt with while implementing the compound application. Analysis of the provenance information is supported by a set of query tools. Not essential, but finally important as well, are management tools to access information within the provenance store and setup/configure the store.

Processes are executed in a distributed environment. Result files, log files, do not provide enough information to reflect the exact history of processes. Interactions between different actors need to be tracked. The provenance system needs to be capable of handling several origins for provenance record submissions, and it should support multiple locations of provenance stores as well.

An effective organizational method to ensure consistency is to avoid redundancy. Records on provenance thus should not contain all the information the application provides in copy. Instead, references to the original data will be embedded into the records. This is achieved by using a *documentation style* with the content referenced. Access to the data itself is provided through a plug-in architecture. By this many types of storage locations may be used by developing appropriate plug-ins.

The integration system is turned into a “provenance enabled” system. In many cases the overhead of provenance recording is neither needed nor desired for its use, thus the fundamental architecture is not changed. Because of this, all data and meta-data for the actual engineering task is stored on the Web-DAV server. Additional information for provenance tracking is managed using the provenance system. Through the mentioned plug-ins all information can be aggregated through specific provenance tools in development for provenance analysis. For the engineering task all information can still be retrieved by means of the integration system.

Unfortunately this forked storage of content may cause inconsistencies due to changes in referenced data. The most effective way to solve this problem would be to store all information in one system only. This approach would be counter productive, as the (remote) provenance store is not designed to cope with high volume data commonly found in the engineering tasks in question. Thus consistency measures have to be ensured on the organizational level.

3.1 Provenance Assertions

To capture the complete process history, three types of p-assertions (assertions on provenance) have been identified [8]: interaction, actor state, and relationship p-assertions.

Interaction p-assertions form the bonding links between components in a process. They capture messages received from or sent by the components or the actors.

As the name implies, the actor state p-assertions provide information on the internal state in the context of a specific interaction. So they may be considered as snapshots of distinguished states of the actor just before or just after it has received a message.

Finally the relationship p-assertions allow relationships between messages and data. They consist of a subject, one/many objects, and a designated relationship

between them. Some examples of possible relationships are as follows: interaction *A is before/after* interaction B; data item *C was zipped to produce* data item D; data item *D is a combination of* data item C and data item B; interaction *A is in reply to* interaction B; interaction *A causes* interaction B. Since, we do not limit what relationships can be expressed, this allows asserting actors to express application specific relationships.

3.2 Provenance Store Access

Access to all interactions with the provenance store are abstracted through standardized WSDL interfaces (see Fig. 2). Enabling provenance awareness in existing applications should be as easy as possible, so recording of p-assertions is handled through a client side library. More complex and interactive functionality of querying the store and managing data inside is provided by external tools. The library supports recording interactions and communication of the components, the user interface, and the data server.

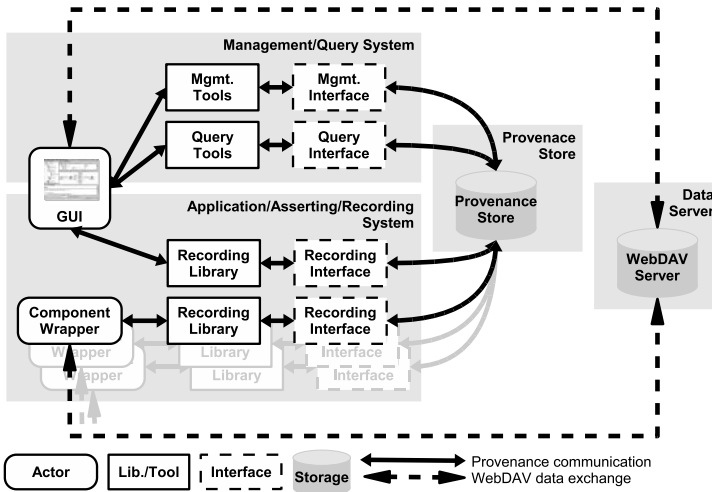


Fig. 2. Access of provenance store and data server by software components

4 Applying Provenance

To implement provenance awareness into the application, and to make use of the recorded provenance information, first the application has to be analyzed methodically. Given that information, the client side library has been implemented in TENT, and p-assertion recording in the appropriate places have been added.

The methodical analysis will be described in the following for a simpler application which implies the same workflow complexity and all required features for provenance recording as the aerospace application.

In the terminology of the application the different functional parts are called workflow *components*. From the point of view of the provenance system, they are called *actors*. Both terms are used as synonyms from different perspectives.

4.1 Implementation for Parameter Study

We are dealing with a workflow (see Fig. 3) that is set up and controlled through the GUI acting as the central process control instance. The data to initialize the simulation will go through some pre-processing to yield a numerical model for a flow simulation (*i1*). A variation component iterates a single parameter. The parameter is inserted into a configuration file that is passed to the simulation (*i2*). During the individual simulation the code writes information on numerical convergence to the standard output stream. This is intercepted and parsed for information to be monitored in the GUI (*m*). At the end of a single simulation step three things happen: The result set is transferred for storage to the data server (*d1*); the result file is sent to the visualization component for direct feedback on the results (*i4*); and finally the parameter variation component is notified to signal the end of the computations (*i3*). When all simulations have been computed, the process control component (GUI) stores unsaved results, the configuration, and the monitoring data on the data server (*d2*). The completed processing of the workflow spanning from *i0* to *d2* defines a complete *simulation*.

As described in Sect. 3, a process's provenance needs to be tracked as events occur. So the different actors need to be provenance aware, and record provenance relevant information making p-assertions. For some problems this may be achieved by simply intercepting the communication between the components. Due to the internal complexity of the different actors, those p-assertions would not document the process properly without further knowledge of the components' internal actions.

Interaction P-Assertions. All interactions between the actors form the linked documentation. Whenever an interaction occurs, an *interaction p-assertion* is submitted. This joins independent actions to a linked process representation. As seen from Fig. 3 the workflow's relevant interactions are *i0* – *i4*, *m1*, *d1*, and *d2*.

As the content for p-assertions for example, *i2* would state the parameter set for the simulation, and *m* would contain the monitoring information.

Additionally *tracers* are introduced here. A tracer is an identifier to tag certain groups of p-assertions within the provenance store (for a subset of a workflow conducted). One tracer for example would identify a complete workflow. It would be introduced with *i0*. Another tracer would be introduced (and changed) with every *i2*, to identify all p-assertions for that specific inner loop.

Actor State P-Assertions. The interactions between actors define the linked application. The state of an actor before or after such interactions may be of importance for a complete provenance representation. To account for this, *actor state p-assertions* have been identified to document an instance snapshot of the actor. All (“boxed”) actors from the figure record their state at various

points of time. For the GUI, this p-assertion may record the workflow's name and configuration. For the simulation, it could be the computation's completion state (converged, crashed, interrupted, etc.).

Relationship P-Assertions. Various causalities within the workflow can be found. They will be recorded through *relationship p-assertions*. Some of these relationships would be as follows: A single interaction $i1$ causes $i2$; an interaction $i2$ causes all of $r2$, $r3$, $r5$, and $r6$; and in a wide spanning relation the initial $i0$ causes the final $d2$.

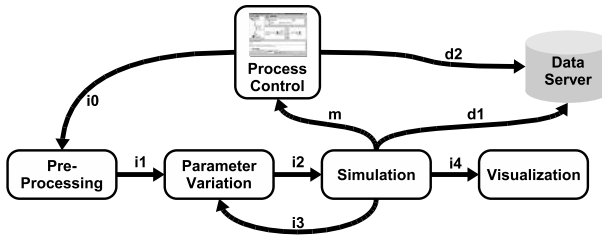


Fig. 3. Mapping of a simple optimization application to provenance architecture

4.2 Aerospace Engineering Application

The implementation for the aerospace application consists of the functional components as described in Sect. 2, with additional actors as the GUI, a coupling manager, a data server, and possibly visualization as shown in Fig. 3. The mapping is performed in the same way as for the parameter study. So it will not be discussed here in further detail.

The aerospace engineering scenario is used for research, and terminally the construction of aircrafts. Specific regulations for reproducibility need to be considered. Thus documentation of conducted processes for legal reasons is important. Additionally the engineer can take advantage of the provenance information, as it will give a better overview and means for analyzing the previously conducted simulations. This may lead to a better insight into similarities and differences previously unnoticed.

This can be obtained by asking questions on the data within the provenance store using the query tool.

- Given some data item, what was the simulation case?
- Given some parameter, in what simulation(s) has it been used?
- What data has been recorded in a simulation with a specific parameter?
- What simulations have been run using a given model (aircraft design)?
- Given two/more simulations with the same setup, what is the result and the difference in provenance?

5 Alternate Approaches

Various approaches for tracking provenance information are analyzed relative to the application discussed.

Embedded within the operating system a provenance enabled file system PASS [9] is developed. For the benefit of not provenance enabling actors within a system, the “observed” information may be very low level and fine granular. The amount of detail can obscure the view onto desired information. Additionally this approach does not capture distributed processes very well, it is not aware of actor states, and thus may not be able to capture the complete process provenance. More interactively CAVE and CODESH [10] facilitate the user with a UNIX like shell. Operations are conducted through it within a session to narrow down the information more usably.

The problem solving environment WinFX has been provenance-enabled by Microsoft [11]. It currently does not allow workflows in inhomogenous platform environments. It provides a closed provenance system, it caches all provenance data during the process, and persists it at process finalization. More open approaches provide the extensible and flexible frameworks Kepler [12] and Karma [13]. Both can be customized flexibly for scientific domains in inhomogenous and distributed environments like TENT. Kepler provides its own provenance tracking system, as does Karma. Additionally Karma has been shown to be able to connect to more generic provenance systems as the ones developed within the PASOA [14] and Grid Provenance projects [1].

6 Conclusion

The need for provenance recording arises in many processes. Reasons for undertaking the necessary steps – that may be considered tedious – can be very different: trust in results, regulations, liability, or just to get better tools for process analysis.

Even though parts of the provenance system are still in development, provenance recording from a smaller scale sample application has been demonstrated already. As the libraries for recording will become usable more easily (due to the ongoing implementation and tools support), it will become easier to implement provenance awareness into the aerospace engineering target application, as well as other possible applications. Furthermore, with availability of the supporting tools and user interfaces for query functionality, inspection possibilities will be provided. These are the necessary tools to start harvesting information available in the provenance store for enhanced insight into engineering processes.

Acknowledgment

This work was a part of the Grid Provenance project, funded by EU IST 511085.

References

1. The Grid Provenance Website. [Online]. Available: <http://www.gridprovenance.org/>
2. Publications of Grid Provenance Project. [Online]. Available: <http://wiki.gridprovenance.org/bin/view/Provenance/ProjectPublications>
3. A. Schütte, G. Einarsson, B. Schöning, T. Alrutz, W. Mönnich, J. Neumann, and J. Heinecke, "Numerical simulation of maneuvering combat aircraft," in *Proc. 14th DGLR Symposium of STAB*, STAB, Ed. Bremen, Germany: Springer-Verlag, nov 2004.
4. T. Gerhold, "The DLR TAU Code – an Overview," in *ODAS 99, ONERA–DLR Aerospace Symposium*, 1999.
5. W. Mönnich and J. J. Buchholz, "SIMULA – Ein Programmpaket für die Simulation dynamischer Systeme," DFVLR, Tech. Rep., 1991.
6. A. Schreiber, "The Integrated Simulation Environment TENT," *Concurrency and Computation: Practice and Experience*, 2002.
7. A. Schreiber, T. Metsch, and H.-P. Kersken, "A Problem Solving Environment for Multidisciplinary Coupled Simulations in Computational Grids," *Future Generation Computer Systems*, 2005.
8. P. Groth, S. Miles, V. Tan, and L. Moreau, "Architecture for Provenance Systems," ECS, University of Southampton, Tech. Rep., oct 2005. [Online]. Available: <http://eprints.ecs.soton.ac.uk/11310/>
9. U. Braun, S. Garfinkel, D. A. Holland, K.-K. Muniswamy-Reddy, and M. I. Seltzer, "Issues in Automatic Provenance Collection," in *Proceedings (unpublished) of the International Provenance and Annotation Workshop (IPAW)*, Chicago, Illinois, USA, May 2006.
10. D. Bourilkov, V. Khandelwal, A. Kulkarni, and S. Totala, "Virtual Logbooks and Collaboration in Science and Software Development," in *Proceedings (unpublished) of the International Provenance and Annotation Workshop (IPAW)*, Chicago, Illinois, USA, May 2006.
11. R. Barga, "Automatic Generation of Workflow Execution Provenance," in *Presentation at the International Provenance and Annotation Workshop (IPAW)*, Chicago, Illinois, USA, May 2006.
12. I. Altintas, O. Barney, and E. Jaeger-Frank, "Provenance Collection Support in the Kepler Scientific Workflow System," in *Proceedings (unpublished) of the International Provenance and Annotation Workshop (IPAW)*, Chicago, Illinois, USA, May 2006.
13. Y. L. Simmhan, B. Plale, D. Gannon, and S. Marru, "Performance Evaluation of the Karma Provenance Framework for Scientific Workflows," in *Proceedings (unpublished) of the International Provenance and Annotation Workshop (IPAW)*, Chicago, Illinois, USA, May 2006.
14. The PASOA Website. [Online]. Available: <http://www.pasoa.org/>

Towards Low Overhead Provenance Tracking in Near Real-Time Stream Filtering*

Nithya N. Vijayakumar and Beth Plale

Department of Computer Science, Indiana University
{nvijayak, plale}@cs.indiana.edu

Abstract. Data streams flowing from the physical environment are as unpredictable as the environment itself. Radars go down, long haul networks drop packets, and readings are corrupted on the wire. Yet the data driven scientific models and data mining algorithms do not necessarily account for the inaccuracies when assimilating the data. Low overhead provenance collection partially solves this problem. We propose a data model and collection model for near real time provenance collection. We define a system architecture for stream provenance tracking and motivate with a real-world application in meteorology forecasting.

1 Introduction

Dynamic data-driven scientific applications utilize data streaming in real-time from environmental sensors and instruments to effect simulation, modeling, and analysis that is more responsive to the physical domain (such as the atmosphere) and the computational environment. Responding in near real time to events in the environment, however, requires minimizing the latency between the occurrence of an event in the environment and the detection and processing of that event through a reduction and analysis pipeline. When decisions are being made in near real time to process incoming data and trigger the appropriate model or service, keeping records of the activities being applied is jettisoned in favor of keeping service time low. Avoiding the recording of historical data is not a viable solution because scientists need the ability to trace a result, such as a statistical result, back to the one or more events in streams that caused them. We need a low overhead model for provenance collection on streams.

The general approach to stream processing is to execute a set of tasks continuously on the incoming events. These tasks can be defined as database queries [1,2,3,11] or a pipeline of computational entities [4] that operate on the data events. We use the term *filters* to refer to the tasks executed on data streams. In this paper we focus on provenance tracking for stream filtering systems. Provenance collection in stream filtering systems pose the following challenges:

1. ***Identifying provenance entities*** - Provenance systems generally collect data about datasets [6]. In a stream filter system, events can be tiny, just a

* This work supported under NSF cooperative agreement ATM-0331480 and DOE DE-FG02-04ER25600.

few kilobytes in size and can be generated at high rates. Collecting provenance on an event by event basis can overwhelm the system. The challenge is in identifying the correct “atomic unit” that is traceable by the stream filter system and strikes a reasonable balance between efficiency and meaningful provenance.

2. ***Capturing stream filtering conditions with low overhead*** - Filters executing on streams can be dynamically deployed and are subject to reconfiguring on the fly [3]. Data streams drop data periodically, such as when a feed from one of the 120 continental U.S. Doppler radars goes down briefly. Network congestion may cause delays and bursts in transfer of stream events. Transport over long haul networks can corrupt data as well. Filters accommodate these changes by changing modes (e.g., approximation mode under imbalanced load [2]). The challenge lies in dynamically and efficiently tracking provenance of filter execution in a distributed environment.
3. ***Maintaining relevance with non-persistent data*** - Filters are typically associated with a lifetime, i.e., they can be specified to run for a particular duration of time [11]. Yet, the products they produce can long outlive the filters that produced them. The challenge is to trace back the source of a derived event and the conditions of stream filtering long after the filtering task was completed.
4. ***Dynamic accuracy estimation*** - Provenance can be used to produce quality of service guarantees. In stream filtering systems, aggregation of the stream data enables the detection of global behavior that cannot be done on single streams. Approximations and accuracy changes in input streams may affect the accuracy of the derived streams [18]. It is challenging to collect provenance across multiple streams and thus deduce the accuracy of derived streams.

In this paper, we address these challenges by creating a data model and a collection model for representing and capturing stream-related provenance that targets the unique needs of stream-driven applications. We define an architecture for stream provenance tracking and show how the provenance collection system fits within the context of the Calder [17] grid-based stream processing system. We demonstrate the feasibility of provenance collection through an example of meteorology forecasting application [5]. We restrict our discussion to provenance collection over data streams that monitor the physical environment, such as the atmosphere, soil, ocean, etc. In future work we will examine the computer-induced streams that support fault tolerance and reliability in a dynamic data driven application.

The remainder of the paper is organized as follows. Section 2 motivates a new model for stream provenance tracking and discusses related work. Section 3 describes the data and collection models for stream provenance tracking. Section 4 discusses an implementation of this provenance model in the Calder [17] system and Section 5 demonstrates an application of this provenance service in meteorology forecasting. Section 6 summarizes the conclusions and outlines the future work.

2 Motivation and Related Work

Currently, provenance techniques do not target stream filtering systems and hence only partially satisfy its needs. In this section, we discuss the challenges identified in Section 1 and related work in this area. We have leveraged on the provenance techniques proposed by the provenance solutions [6,14,9,18], in creating a new provenance tracking model for streams. A recent survey [12] provides an overview of the existing provenance solutions for e-Science applications.

The first challenge is to identify the smallest unit for which provenance is collected in stream filtering systems. A data stream is an indefinite sequence of time ordered events, $(e_1, e_2, \dots, e_n, e_{n+1})$ where $timestamp(e_n) < timestamp(e_{n+1})$. The virtual data schema used by the Virtual Data Grid project [6] represents the datasets, their relationships and the computations. In streams, a dataset corresponds to an event in a stream, so tracking provenance of datasets without burdening the system is a challenge. Provenance information can be encoded along with each dataset, but the events are not persistent and hence the provenance information is not available to the stream filter system after the event is processed. We would like to trace the source of events in derived streams to the events of inputs streams without identifying every event individually.

The second challenge is in capturing the provenance history of streams and filter states with low overhead on the system. Response time is crucial for stream filtering systems. Thus, low overhead (in time, resources etc) for provenance collection is of utmost importance. Log4j [8], which logs error and status messages to a log file, for instance creates a non-trivial load on the service about which it records data. Also aggregating the provenance traces into meaningful information is difficult with systems like Log4j. Capturing the provenance of filter execution is internal to the system and is expected to have a lower overhead compared to provenance collection for data streams.

The third challenge is to trace the source of a stream long after the filtering process has completed. We need to be able to clearly identify the environment in which a particular set of events (subset of a stream identified by timestamp) were generated. Also stream filtering systems adapt themselves to changes in underlying resources [2,11]. This involves changes in query execution plans and approximations when streams are not available. The provenance model needs to accommodate these changes and preserve the details for future reference.

The PASOA [14] project focuses on collecting provenance information on interaction between services in a workflow using a formalized protocol [7]. Karma provenance service [13] is used in the LEAD [5] project for tracking provenance of meteorological data and its usage in web-services. In stream filtering systems, the provenance is collected for each stream and the filters that execute on the streams. The communication between the internal components of the stream filtering system is not as important as the entities as a whole. Security issues in a SOA-based provenance system is discussed in [16]. Security of provenance data is an important issue that is applicable to streams as well.

Finally, the provenance model needs to enable tracing the accuracy of a subset of the stream to a specific time period. Deducing an accuracy value for an

derived event based on the accuracy of the input streams and stream filtering environment is a challenge in itself. Trio [18], is a database system that augments conventional data model with accuracy and lineage and enables querying using understandable extensions to SQL. This is made possible by associating lineage and accuracy information with datasets in Trio. This is not applicable to streams, because the events are not persistent. We need to be able to trace accuracy of a subset of stream long after the stream was generated.

3 Provenance Tracking in Stream Systems

We propose the following data model and provenance collection model for stream filtering systems to address the challenges discussed in previous sections.

Data Model

We identify three atomic units of provenance collection in streams: *base streams*, *adaptive filters* and *derived streams*. *Base streams* are streams that are generated outside the stream filtering system. The generation source may be a instrument, experiment, or any process. *Adaptive filters* are declarative queries or application code that are associated with a life time and continuously execute on the data streams; *Derived streams* are streams that are produced by executing adaptive filters on base streams or other derived streams.

We propose a timestamp based *append only stack* approach for collecting provenance of streams and filters, and a *bottom-up provenance tree* to associate the base streams and derived streams. By append only stack we mean a data structure in which information can only be added not removed; and also that the latest information identified by the timestamp represents the current status. This provenance stack accommodates a set of information collected initially (base provenance information like the input streams, filter used etc) and a list of changes (dynamic provenance information like changes in stream rates, filter mode changes etc). The provenance stack can to be stored as a file or as a table in a database, in a persistent manner. Provenance information for a base stream constitutes the data format of the stream, its sources, information on the stream generation process, owner and permissions, user defined annotations and metadata. When users specify the filter to be executed, it could be appended with some annotation on its purpose. The execution plan and annotations serve as base provenance information for the filters. For a derived stream, the base provenance information is the list of input streams and the filters executed to derive the stream. The derived streams refer to the provenance of their input streams and that of their filters. Thus the lineage of a derived stream can be traced using a provenance tree where the input streams are at the root level and derived stream is at the leaf (bottom-up provenance tree).

System metadata (owner, permissions, etc) and user defined metadata, can be stored as name value pairs or using predefined schemas. The model supports annotations by storing them as inline text or storing them independently as in [9], and referring to them using URLs.

Low Overhead Provenance Collection Model

Our collection model is based on the assumption that each stream consists of a sequence of time ordered events and each event is associated with a timestamp. A subset of stream can be defined by a starting timestamp and an ending timestamp. Data streams are subject to rate and accuracy changes. To store the provenance of a stream, it is important to capture the changes in input streams and associate them to the set of events in the derived stream that are affected by the change. To facilitate this, the changes are logged with a starting timestamp. It is sufficient to match a change with the starting timestamp as it applies to the rest of the stream from that timestamp onwards. This model helps to capture the provenance information of a stream dynamically and keep the information up-to-date.

We need to be able to trace the source of a derived event and the stream filtering environment under which it was produced, long after the filtering task is completed. For this we only need to store the provenance history of all streams and filters in a persistent storage and not the events themselves. After the base provenance collection, information is logged only when something changes in the environment and hence the overhead for provenance collection and need for storage space are minimal. The provenance history is stored as one provenance stack per stream or filter. By looking through the provenance information of the derived stream, their input streams and the filters, one can trace the filter states and conditions under which a particular set of events were derived. The same methodology applies to deducing the accuracy of the derived stream events. Given, that a well-defined formula exists in a particular domain to calculate the accuracy of the derived stream from that of the input streams and the filter being used, the provenance model can support it.

Example - A sample provenance document for a derived stream is given in Figure 1. The stream under consideration is a derived stream (ID D0010) and uses two input streams (base stream with ID B0011 and derived stream with ID D0005). The stream was started at Feb-10-2006 at 13:00:00 hours and the steady rate was 50 events/sec recorded 15 minutes after stream was registered. From the change log, we can see that the stream B0011 was missing for about 10 minutes during which the filters changed mode to approximate the missing stream. The accuracy of the derived stream reduced to 85% during the approximation. From the next change log timestamped at 13:45:00 it is known that the B0011 stream came back up and hence the approximations were removed and the accuracy of the derived stream increased to 100%.

4 Calder Provenance Service

Calder [17], is a distributed stream processing system that supports a service interface for query processing. The Calder system, is an extension of the dQUOB [11] project, and is composed of two subparts: a set of data management services and a set of dynamically configurable query processing engines,

```

<derivedstream>
  <name>Temperature Feed</name>
  <uniqueID>D0010</uniqueID>
  <queryID>Q0099</queryID>
  <inputstreams>
    <streamID>B0011</streamID>
    <streamID>D0005</streamID>
  </inputstreams>
  <systemmetadata>
    <name> owner </name> <value> foo </value>
    <name> permissions </name> <value> open to everyone </value>
  </systemmetadata>
  <starttime> <timestamp> 13:00:00 Feb-10-2006 </timestamp></starttime>
  <changelog>
    <event>
      <timestamp> 13:15:00 Feb-10-2006 </timestamp>
      <rate> 50 events/sec </rate>
    </event>
    <event>
      <timestamp> 13:34:56 Feb-10-2006 </timestamp>
      <description> B0011 down</description>
      <approximation> Sampling </approximation><accuracy> 0.85 </accuracy>
    </event>
    <event>
      <timestamp> 13:45:00 Feb-10-2006 </timestamp>
      <description> B0011 up</description>
      <approximation> None </approximation><accuracy> 1.0 </accuracy>
    </event>
  </changelog>
</derivedstream>

```

Fig. 1. Sample Provenance Document for a Derived Stream

as shown in Figure 2. Filters are specified as database queries expressed using a subset of SQL. Calder uses an extended OGSA-DAI v 6 [10] grid data service interface to support a data stream resource. The provenance service of Calder implements the provenance models described in Section 3 and supports a service oriented interface for querying the provenance information. It uses a native XML database to store the provenance history of streams and filters. Users register the base streams and filter queries by invoking the provenance service. Registration of derived stream is made by the system when a new query is submitted. The derived streams can then be retrieved in a timely manner as streams or asynchronously as chunks from the rowset service (Figure 2). Once a stream/filter query is registered, users can append it's provenance stack with additional information like annotations and metadata.

We appended the Calder system with a monitoring service to facilitate dynamic collection of provenance information during stream processing. The provenance service interfaces with the monitoring service by an event-notification interface. Figure 2 shows the architecture of Calder with provenance and monitoring services. The provenance information propagates as shown in Figure 3. The query planner is responsible for executing the filtering query. It updates the monitoring service whenever the execution plan of a query changes. A query execution plan may change due to a set of streams going down or a processing node failure. The monitoring service also gets updated by the query processing engines when an event of interest occurs. Events of interest include rate changes, missing streams, approximations and accuracy changes. The flexible service interface of Calder enables a scalable framework. A single instantiation of the Calder system

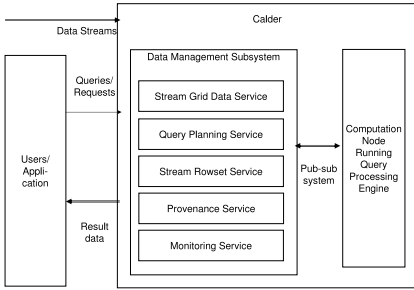


Fig. 2. Calder Architecture

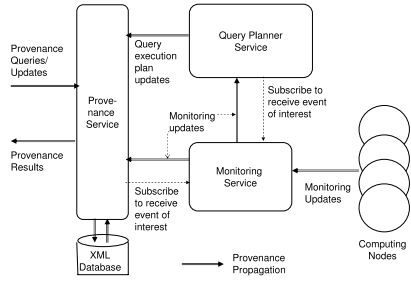


Fig. 3. Provenance Updates in Calder

can spawn multiple internal services and query processor engines on-demand to accommodate the load.

5 Application in LEAD

We discuss an application of Calder and its provenance service in the context of the LEAD [5] meteorology forecasting project. *Time bounded stream mining* is the dynamic deployment of data mining agents that run for a bounded period of time, look for environmental events of interest, and report their results in the form of a trigger that can be used to invoke subsequent behavior. Our approach to on-demand data mining is to view the streams generated by heterogeneous instruments as belonging to a single data domain over which processing can be performed. The user interacts with the data domain through a declarative query.

The value to the user of time bounded stream mining can best be illustrated by means of an example. An atmospheric scientist is studying spring severe weather over the US Midwest. When a large storm front is moving in from across the Plains, she wishes to kick off a small, regional forecast simulation wherever storm cells emerge. She does this by sprinkling data mining agents in front of the storm line, each configured to run for a specified time (say 3 hours). Each mining agent executes a tight loop for the specified time looking at NEXRAD Level II Doppler scans within a small geospatial region for severe storm precursors. If the agent finds something of interest, it triggers a regional (small scale) forecast prediction simulation. When complete, the simulation invokes statistical analysis on the results. Figure 5 shows a sample continuous filtering query that can be executed on the incoming Level II data for the given scenario. Calder dynamically instantiates the query and query processing engine on a remote node, say on the TeraGrid [15]. The provenance service collects information on each of the data mining agents and streams used. It tracks changes in stream rates, temporary outages if any, and the conditions under which the operation was conducted. Such provenance information can be used to trace the result of a forecast model back to one or more events in streams that caused them.

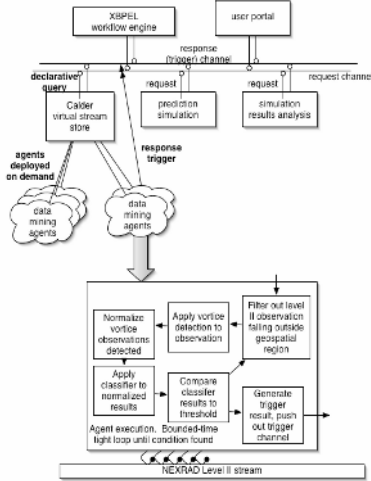


Fig. 4. Data Mining Agents in LEAD

RULE C:1

```

SELECT FROM level II as scan
WHERE
  (classif (normalize(
    MDA (scan in "bounding box")))
    > "threshold"
START 2006-04-25 12:00:00
EXPIRE 2006-04-25 20:00:00
THEN
  ACTION (threshold, scan)

```

Fig. 5. Query: Run the given data mining algorithm on incoming data streams and do the specified action

6 Conclusion

This paper discusses the unique challenges of low-overhead provenance collection in stream filtering applications. We introduce data and collection models addressing these challenges and discuss a prototype implemented as part of the Calder stream processing system. Our work is motivated through a real-world application of meteorology forecasting. Our current effort is focused on evaluating the performance of the system in the context of the dynamic weather forecasting. We are also working on extending the provenance collection service into a larger context management service that provides usage patterns, history of streams, user annotations and feedback on quality of streams.

References

1. Abadi, D. J. et. al.: The Design of the Borealis Stream Processing Engine. Conference on Innovative Data Systems Research (CIDR) (2005)
2. Babcock, B., Babu, S., Datar, M., Motwani, R., and Widom, J.: Models and issues in data stream systems, ACM Symposium on Principles of Database Systems, (2002)
3. Chandrasekaran, S., et. al.: TelegraphCQ: continuous dataflow processing. International conference on Management of Data (SIGMOD) (2003).
4. Chen, L., Reddy, K., and Agrawal, G.: GATES: A Grid-Based Middleware for Processing Distributed Data Streams. IEEE International Symposium on High-Performance Distributed Computing, (2004)
5. Drogemeier, K., et al.: Service-oriented environments in research and education for dynamically interacting with mesoscale weather. IEEE Computing in Science and Engineering, (2005), 7(6)

6. Foster, I., Vockler, J., Wilde, M., and Zhao, Y.: The Virtual Data Grid: A new model and architecture for data-intensive collaboration. Conference on Innovative Data Systems Research, (2003).
7. Groth, P., Luck, M., and Moreau, L.: A protocol for recording provenance in service-oriented Grids. International Conference on Principles of Distributed Systems (2004).
8. Log4j. Apache Software Foundation. <http://logging.apache.org/log4j/>
9. Myers, J.D., Chappell, A., Elder M., Geist A., and Schwidder, J.: Re-Integrating the Research Record. IEEE Computing in Science and Engineering (2003) 5(3):44-50.
10. The OGSA-DAI Project. <http://www.ogsadai.org.uk/>
11. Plale, B., Schwan, K.: Dynamic querying of streaming data with the dQUOB system. IEEE Transactions on Parallel and Distributed Systems, 14(4):422-432, (2003).
12. Simmhan, Y. L., Plale, B., and Gannon, D.: A survey of data provenance in e-science. ACM SIGMOD Record, (2005) 34(3):31-36.
13. Simmhan, L. Yogesh, Plale, B., Gannon, D., and Marru, S.: Performance Evaluation of the Karma Provenance Framework for Scientific Workflows. Intl Provenance and Annotation Workshop (2006).
14. Szomszor, M., and Moreau, L.: Recording and reasoning over data provenance in web and grid services. Int. conference on ontologies, databases and applications of semantics (2003).
15. TeraGrid. <http://www.teragrid.org>.
16. Tan, V., Groth, P., Miles, S., Jiang, S., Munroe, S., Tsasakou, S., and Moreau, L.: Security Issues in a SOA-based Provenance System. Intl Provenance and Annotation Workshop (2006).
17. Vijayakumar, N., Liu, Y., Plale, B.: Calder query grid service: Insights and experimental evaluation. To appear in CCGrid (2006).
18. Widom, J.: Trio: A system for integrated management of data, accuracy, and lineage. Conference on Innovative Data Systems Research (2005).

Enabling Provenance on Large Scale e-Science Applications

Miguel Branco^{1,2} and Luc Moreau²

¹ CERN, European Organization for
Nuclear Research, CH-1201 Genève
Miguel.Branco@cern.ch

² University of Southampton,
Southampton SO17 1BJ, United Kingdom
L.Moreau@ecs.soton.ac.uk

Abstract. Large-scale e-Science experiments present unprecedented data handling requirements with their multi-petabyte data storages. Complex software applications, such as the ATLAS High Energy Physics experiment at CERN, run throughout Grid computing sites around the world in a distributed environment, with scientists performing concurrent analysis on data and producing new data products shared among the collaboration. In this paper, we introduce a multi-phase infrastructure to achieve data provenance for an e-Science experiment. We propose an infrastructure to integrate provenance onto an existing legacy application with strong emphasis on scalability and explore the relationship between provenance and metadata introducing a model where data provenance is made available as metadata through a separate reasoning phase.

1 Introduction

Large-scale e-Science experiments are underpinned by an iterative e-scientific process by which data are produced, made available to the community, analyzed, reproduced, verified, curated and shared. The success of e-Science experiments, or computationally intensive sciences carried out in highly distributed network environments is often defined by the ability to efficiently analyze data. An aspect to this efficiency is the ability to understand the *origin of data*, which is usually referred to as *provenance*, under analysis: whether it is raw data as received from the detector or the result of possibly very complex computations. Consider the ATLAS detector for the Large Hadron Collider [14] which will deliver hundreds of megabytes of raw data per second when online. This data is expected to be available for studies for up to twenty years. Scientists will be located in their institutes anywhere in the world, analyzing data in isolation or as part of a small group or a larger community, possibly spanning multiple institutions. Their analysis will be undertaken concurrently on data, producing derived data products that may or may not be relevant to others, shared or validated.

In this paper, we present a novel provenance infrastructure suitable for large-scale applications, particularly in how it copes with storage and handling of large data volumes. We present a design with a novel reasoning phase enabling provenance to be

made available as metadata. In addition, we introduce an approach for ease of integration of provenance onto existing legacy applications. The ATLAS High Energy Physics experiment was used as a scenario for this work.

We start by presenting the constraints for integrating provenance onto large-scale e-Science experiments, then introduce our data provenance definition and present the provenance infrastructure. Finally, we present related work and conclude.

2 Provenance for Large-Scale e-Science Experiments

Provenance is defined, in the Oxford English Dictionary, as *the place of origin of something, or the fact of something coming from a particular source*. It is our goal to provide *e-scientists* with the tools to track origin of data, or *data provenance*, in their experiments.

We start by defining broadly the environment and constraints on which we base our work. Creating a model involves foremost understanding the restrictions of the domain. As such, when modeling a provenance infrastructure, we took into account the requirements and constraints typical of High Energy Physics.

We aimed to ease integration of a provenance infrastructure with the experiment's existing software infrastructure. Indeed, proposing an infrastructure that requires dramatic changes to the software of existing e-Science experiments in order to support *data provenance* might certainly be a useful exercise but would prevent widespread adoption, given the extensive legacy code base. Hence, grappling with the legacy challenge – integration with existing systems – is a core concern from start when defining, modeling and implementing the infrastructure.

Therefore, we have identified a small set of restrictions, which we believe are common to many situations involving integration with a large-scale e-Science experiment. These are: the inability to alter the majority of (deployed) software components; difficulty of layering new middleware across the application; diversity of design practices across existing software, as software components may not follow a single architectural style; usage of complex workflows with interwined interactions, difficult to identify clearly; the presence of a highly distributed environment for producing and analyzing data as well as for developing software.

Another restriction is the difficulty in understanding the very complex workflows in production. As an example, a single ATLAS workflow involves several gigabytes of software libraries, many remote accesses to databases (detector conditions, calibration, ...) and the set of end-user configurable options only within the ATLAS analysis framework are close to a thousand. The execution of the workflow is also a significant computing effort, run on the Grid and partitioned onto multiple tasks executed in parallel throughout computing sites around the world.

With the previous restrictions in mind we now define *provenance*. First we define a documentation record as being the view of an intervenient actor (a client or a service) on part of a process execution. We define *data provenance properties* as the relevant set of descriptive properties of the data products for end users. We also define *reasoning* as the derivation of data provenance properties involved in the execution, by applying a reasoning algorithm to a set of documentation records. Finally, we define

provenance as the end result of applying context-specific reasoning over a set of records that document the execution of a process, with the goal of deriving a set of properties of the data products involved in the execution.

An important factor of our definition is that provenance is to be determined in relation to the data and not meant to determine e.g. the provenance of a workflow execution. Our goal is ultimately to allow users to understand the origin of a piece of data by looking at the derived set of provenance properties for the data product under analysis.

3 Creating Provenance-Aware Applications

Provenance-aware applications are defined as applications that implement our provenance infrastructure. Legacy e-Science experiments are defined as complex, large-scale experiments with an existing code base, not designed from start to provide provenance for its end-users.

We start by introducing an important assumption regarding service-orientation architectural style and then present our provenance infrastructure.

3.1 Why Service-Oriented Architectural Style is Useful

We take the view that complex software applications are typically designed using a service-oriented approach [11] or can easily be modeled using service-orientation principles. Integration with large-scale legacy applications is facilitated if coupling the provenance infrastructure has minimal interference with the existing application. The flexibility of the service-oriented model allows for relative ease of integration with legacy systems (e.g. by identifying and wrapping the various legacy components as services). This was seen in our analysis of the ATLAS Experiment where the various components involved with the ATLAS Production System followed diverse architectural styles but were easily modeled as coarse-grained services with simple interactions.

The service-oriented architectural (SOA) style can be defined as having actors (services) which interact with one other by exchanging messages, following a service description. Deciding which components, sub-set of components or composition of components should be a service in SOA is a multi-dimensional problem exceeding the scope of our work. For what concerns our infrastructure, the finer-grained the services the greater is the potential understanding of the data flow, if we consider our infrastructure relies on a protocol (presented below) that captures message exchanges between services. Nonetheless having finer-grained services may lead to potentially recording more information that may be redundant, unimportant and expensive to record, store or analyze.

Alternate approaches to provenance often create new data and workflow modeling languages that enable provenance of data to be fully determined, even mathematically proving the models as complete. These approaches are especially common in the context of relational databases [1,3,4]. While these works provide an important foundation, the legacy challenge – integration into existing systems, particularly given by the requirement to apply new modeling languages – prohibits its applicability in some

contexts. It is precisely these scenarios that we intend to cover by providing a model that impose little restrictions and changes to existing systems.

3.2 A Provenance Infrastructure

To create provenance-aware applications, we have defined a model consisting of four phases: creating documentation, storing, reasoning and querying. The overall model is shown on Figure 1.

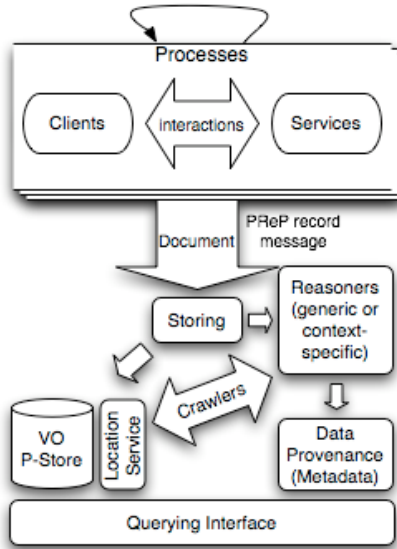


Fig. 1. Logical model for a provenance infrastructure

Documenting a Process. Documenting the execution of a process translates into creating documentation records, as shown on Figure 1. This is done in our infrastructure using PReP [9,10] which also defines a provenance store as the repository for the documentation records. PReP defines a representation of process documentation suitable for service-oriented architectures, introducing a generic protocol for recording provenance. An element of process documentation is called a p-assertion: an assertion that is made by one of the intervening actors and pertains to a process. In PReP, actors (clients and services) involved in a process may provide their view on an interaction. The flow of the process is captured by relating several assertions, along with the corresponding clients and services.

The documentation records conform to a PReP schema. PReP does not limit the scope of the assertions made by the intervenient actors – an actor may assert any information it wishes, as long as it pertains to the process - but in our usage of PReP we assume actors assert a minimal set of assertions (typically the message exchanges and the request-response relationship between messages) to facilitate integration with

legacy applications – that is, not to alter significantly its existing and deployed services, but by only wrapping the external interfaces to support PReP.

Storing Documentation Records for a Process. Once documentation records have been created, they are stored and an index must be created to efficiently locate them. The *storing* phase implements the storage of documentation records.

PReP has introduced the concept of a provenance store (p-store), as the repository for documentation records. It is expected that documentation records may themselves become a significant portion of the total data volume for large-scale experiments such as ATLAS. Therefore in our infrastructure we have extended PReP's p-store concept and defined that documentation records may be replicated onto multiple physical instances. Thus what PReP defines as a provenance store is for us a logically unique provenance store, which may in our model have multiple physical instances. We have defined the existence of a *location* service providing the single entry point to locate documentation records (given a unique identifier) across multiple instances of the same logical repository. The documentation records are write-once-read-many, meaning that contents do not change after recording. This property allows for straightforward replication of data without concerns for concurrency. In addition our design foresees (parts of) a physical instance of a p-store to be serialized on disk, and shipped around computing sites for improved accessibility or archival. Also, we defined the storage layer as providing secure access to documentation records by implementing Grid-enabled authentication and authorization.

Reasoning over Documentation Records. Reasoning consists of analyzing documentation records and extracting a set of data provenance properties for the data elements involved in the workflow execution, as required by a reasoning algorithm. This set of properties constitutes what we refer to in our model as the *data provenance* for a data product. The definition implies that data provenance is overall dependant of each particular reasoning algorithm. Often, reasoning algorithms depend on the context (e.g. for ATLAS, 'luminosity' may be considered part of the data provenance for certain categories of event data). Others are generic even across experiments from different scientific domains (e.g. the software versions used during data processing).

Reasoning is typically done asynchronously in relation to the recording or storage of documentation records: it begins after documentation records have been stored. To reason over documentation records, the records must first be found and (parts of) its contents retrieved. The generic *location* service, shown on Figure 1, allows physical locations for documentation records to be resolved, typically by giving to the location service a globally unique identifier. A reasoning algorithm may require more than a single documentation record as input. Therefore it is usually necessary to find a set of related documentation records, according to some strategy that is dependant of the reasoning being applied. A *crawler* is the component from our infrastructure that can navigate throughout the contents of the provenance repositories (the *p-store* shown on Figure 1), finding physical copies of the required documentation records. The crawler takes into account security policies for accessibility to the documentation records. Also, in experiments with a large set of documentation records, typically only a sub-group of those records can be analyzed at any given time. Restrictions range from time available for reasoning to the accessibility of stored documentation records (e.g., for ATLAS some documentation records may be archived onto tape which leads to a prohibitive

access time for retrieval). When crawlers are actively navigating the documentation records they may detect that certain records are not available, either permanently or temporarily, and feed this information back to the reasoning algorithm.

Querying Newly Defined Data Provenance as Metadata. Ultimately, the goal is to provide the output from the previously described reasoning phase to end users. This is accomplished by defining new metadata attributes, which we referred to as *properties*, associated with the original data products.

Reasoners operate on the documentation records to produce metadata that is kept on a metadata catalog. User queries are then directed to the metadata catalog avoiding the need for end-users to directly query the p-store. This two-phase model is designed to cope with large volumes of data in a scalable manner. As a side effect, the provenance infrastructure becomes a provider of metadata for the application. We call *tagging* of data to the association of data with its data provenance. Tags also serve as indexes to the data products allowing faster and user-friendlier understanding of the data.

Motivation for a Reasoning Phase. One important feature of our infrastructure is that end-users do not directly query over the documentation records contained within the provenance store, but only the metadata catalog. In the next paragraphs, we present a few motivations for this choice.

When analyzing a set of documentation records there is no guarantee that the information contained is consistent, as the documentation records may be incomplete (e.g. the process may have not been fully documented or parts of the p-store may have got lost due to a storage failure). This is particularly relevant for experiments with long lasting workflows, such as the ATLAS experiment, where producing a single dataset (single workflow) may last several months and be computed and stored throughout tens of computing sites in parallel. In some scenarios, incomplete information may still be sufficient to deduce data provenance for a piece of data, e.g. for ATLAS, if the majority of the data products part of a dataset follow a certain software version and the dataset was assigned to be processed at a single computing site, it is safe to assume that all dataset partitions follow the same software version even though the documentation records for each individual partition may not be known. This type of statistical significance is very much dependent on the particular data provenance property being derived (in the example: ‘*software version*’ for a ‘*dataset*’).

Another important motivation comes again from the large volumes of data we expect to handle. For ATLAS, not all documentation records may be accessible at all times (e.g. parts of the provenance store may be archived onto tape) so it is up to the implementers of the reasoner algorithm to take into account a set of context-specific assumptions and decide how the reasoning infrastructure should react.

In addition, end-users should not be given the ability to directly query documentation records, as their queries may cause large sets of the p-store to be read. This is particularly important considering the provenance store may spread many sites with different performance for data access. Finally, in ATLAS, we have identified a large set of provenance queries, which are repeatedly executed by many users (e.g. “what is the set of algorithm versions used to produce dataset X?”), requiring large portions of

the p-store to be read in order to derive the necessary provenance property. Formalizing the reasoning step so that the query to the p-store is performed only once by the infrastructure and in the most optimal access conditions to the documentation records saves many unnecessary queries, improving scalability and manageability of the store.

4 Related Work

Buneman et al [1] have presented models for determining data provenance (or lineage, or pedigree) in the context of relational databases. A technique was also presented on [2] for optimizing archival of data based on recording semantic continuity of elements, introducing two definitions: "where-provenance" - determining where a piece of data came from and "why-provenance". Cui and Wisdom [3] further studied "why-provenance" that can be defined as: "why is a certain piece of data in the database" and "what tuples in a database D contributed to some piece of data d in query Q(D)". All the work described so far has been developed on the context of relational databases, although authors claim is applicable elsewhere. [4] presents techniques to understand provenance in the context of data warehouse systems. Silva et al [6] present a model for "knowledge provenance" including proof-like information on how a question-answering system arrived at the response. Woodruff and Stonebraker [7] argue for fine-grained data lineage and present a method based on an inverse function as the methodology to determine provenance. All these models were built using an idealized, close world of databases. In this world many natural restrictions and simplifications apply and query languages are often created to satisfy the underlying data models (when new data models are not defined). The data warehouse concept cannot apply directly since there are many concurrent, possibly inconsistent, repositories available worldwide. Notions such as "distributed data", "distributed applications" have to be first-class "citizens" of our provenance model. Similarly ad-hoc approaches to provenance, such as electronic logs, can hardly cope with modern day e-Science data demands. Electronic logs are typically updated by manual user input or by simple scripts. In addition, e-logs do not necessarily have any associated reliability in the information they contain since it is mostly updated by humans. Zhao et al. [13] the provenance logs are extended by creating annotations based on concepts drawn from an ontology. This allows records to be linked with each other by means of inference over the associated concepts. COHSE (Conceptual Open Hypermedia Services Environment) integrates the ontology service with an annotation and linking service.

Chimera virtual data system [8] is based on the assumption that explicit representation of computational procedures used to derive data can be defined, enabling on-demand data generation from analyzing the expressive representation of computational procedures.

Szomszor and Moreau [5] present a model for recording and retrieving provenance on large-scale, dynamic and open environments such as those of Grids [12] and Web services where a workflow enactment engine is responsible for interacting with the VO services and is altered to submit information to a provenance repository.

This work is distinguishable from PReP [9,10] in that the former is a provenance recording protocol. We build upon this protocol, detailing the storage handling, and defining an infrastructure to reason over PReP's documentation records systematically

exposing data provenance as metadata, allowing developers to asynchronously build reasoning algorithms and providing end-users with a simplified query interface based on a metadata representation.

Finally, large-scale legacy e-Science experiments require flexible methodologies to handle the processing, publishing, analysis, verification and curation of data and none of these alternate approaches was particularly designed to take into account the naturally distributed environment on which these experiments work.

5 Conclusion

The proposed provenance infrastructure enables provenance-awareness for large-scale e-Science experiments, particularly those handling large volumes of data. The infrastructure allows for provenance recording, storage, reasoning and querying by identifying the multiple stages for provenance integration in an application. It builds upon PReP by specifying how the provenance recording protocol can be integrated with a legacy application, handling large volumes of data in the provenance store and with a separate reasoning phase. Asynchronous reasoning algorithms allow for a flexible and scalable framework for data provenance whose representation relies on a metadata catalog. In terms of future work we intend to continue prototyping the model identified in this paper and evaluate its applicability on a set of scenarios in the context of the ATLAS Experiment.

References

1. P. Buneman, S. Khanna, and W.C. Tan. Data provenance: Some basic issues. In *Foundations of Software Technology and Theoretical Computer Science*, 2000.
2. P. Buneman, S. Khanna, K.Tajima, and W.C. Tan. Archiving scientific data. In *Proc. of the 2002 ACM SIGMOD International Conference on Management of Data*, pages 1–12. ACM Press, 2002. ISBN 1-58113-497-5.
3. Y. Cui and J. Widom. Practical lineage tracing in data warehouses. In *Proceedings of the 16th International Conference on Data Engineering (ICDE'00)*, San Diego, California, February 2000.
4. J. Widom Y. Cui. Lineage tracing for general data warehouse transformations. In *The VLDB Journal*, pages 471–480, 2001.
5. M. Szomszor and L. Moreau. Recording and reasoning over data provenance in web and grid services. In *International Conference on Ontologies, Databases and Applications of SEMantics (ODBASE'03)*, volume 2888 of *Lecture Notes in Computer Science*, pages 603–620, 2003.
6. R. McCool P. Silva, D. McGuinness. Knowledge provenance infrastructure. *IEEE Data Eng. Bul l.*, 26(4):26–32, 2003.
7. Woodruff and M. Stonebraker. Supporting fine-grained data lineage in a database visualization environment. In *ICDE '97: Proceedings of the Thirteenth International Conference on Data Engineering*, pages 91–102, Washington, DC, USA, 1997. IEEE Computer Society. ISBN 0-8186-7807-0.
8. I. Foster, J. Voeckler, M. Wilde, and Y. Zhao. Chimera: A virtual data system for representing, querying, and automating data derivation, 2002.

9. P. Groth, M. Luck, and L. Moreau. Formalising a protocol for recording provenance in grids. In Proc. of the UK OST e-Science second All Hands Meeting 2004 (AHM'04), Nottingham, UK, September 2004.
10. P. Groth, M. Luck, and L. Moreau. A protocol for recording provenance in service-oriented grids. In Proceedings of the 8th International Conference on Principles of Distributed Systems (OPODIS'04), Grenoble, France, December 2004.
11. Munindar P. Singh and Michael N. Huhns. Service-Oriented Computing: Semantics, Processes, Agents. John Wiley & Sons, Ltd., 2005.
12. I. Foster, C. Kesselman, and S. Tuecke. The anatomy of the grid: Enabling scalable virtual organizations. Lecture Notes in Computer Science, 2001.
13. Jun Zhao, Carole Goble, Robert Stevens and Sean Bechhofer. Semantically Linking and Browsing Provenance Logs for e-Science. In International Conference on Semantics of a Networked World Revised Selected papers, Springer LNCS 3226 pp 157-174, Paris, France, June 17 -19, 2004.
14. ATLAS Computing Group, ATLAS Computing Technical Design Report, <http://doc.cern.ch/archive/electronic/cern/preprints/lhcc/public/lhcc-2005-022.pdf>, June 20, 2005

Harvesting RDF Triples

Joe Futrelle

National Center for Supercomputing Applications
1205 W. Clark St., Urbana IL 61801, US
futrelle@uiuc.edu

Abstract. Managing scientific data requires tools that can track complex provenance information about digital resources and workflows. RDF triples are a convenient abstraction for combining independently-generated factual statements, including statements about provenance[1]. Harvesting is a strategy for asynchronously acquiring distributed information for the purposes of aggregation and analysis[2]. Harvesting typically requires that information be temporally scoped and attributed to some creator or information source. An RDF triple asserts a fact without attributing it to any actor or period of time, so the abstraction must be extended to support typical harvesting scenarios. This paper compares standard, conventional, and non-standard means of extending RDF triples to associate them with attribution and timing information. Then, it considers the implications of these techniques for harvesting and presents some implementation sketches based on a journaling strategy.

1 Introduction

In NCSA's Cyberenvironments project (<http://www.ncsa.uiuc.edu/Projects/>), the need to capture provenance from multiple, distributed, heterogeneous portal and workflow tools has led to the development of an RDF harvesting strategy. Because it is impractical to retool every aspect of the complex technical infrastructure used in science to support RDF API's, tools, and implementations, our approach is to wrap these tools in middleware that makes minimal assumptions about its environment, building consensus around simple abstractions such as log files and journals. The combination of a bottom-up implementation strategy with RDF's descriptive power has given our social networking and data mining efforts diverse new sources of provenance information with relatively little investment in new frameworks, protocols, and API's.

2 Conceptual Overview

Facts and Contradiction. Resolving contradictions across multiple, independently-produced RDF graphs requires rules that sometimes depend on second-order descriptions. For instance, imagine a reasoner that resolves contradictions between pairs of statements by ordering them alphabetically and rolling loaded dice. The dice-loading parameters are "global" in that they are

independent of the statement pairs. Now imagine a reasoner that resolves contradictions between pairs of statements by selecting the statement that was made by the most trusted actor. In that case the trust parameters are “local” in that they are derived in part from second-order information about which actor made which statement[3].

Actors and Statements. Actors in a distributed environment produce and consume information. Since it often matters which actor produced which information, representing the information produced by actors as triples is insufficient to enable reasoning about the information, since the triples alone do not contain the context required to parameterize rules that depend on who said what. If actors can be uniquely identified, managing that contextual information amounts to associating an actor’s identity with each triple or set of triples. Contradictions can then be resolved in any number of ways, e.g., trust ranking.

Time and Negation. In general, models change over time[4]. RDF has no standard means to scope assertions temporally, which would enable contradictions to be resolved based on sequencing and other temporal logic. Time is an especially useful kind of contextual information, because it is “global” and therefore does not require coordination between data producers beyond clock synchronization, a solved problem[5].

3 Representation

RDF reification provides a standard way of associating arbitrary contextual information with any triple[6], including attribution and timing. However, representing attribution and temporal information about triples *efficiently* requires constructs that aren’t available in RDF.

Reification. Second-order descriptions in RDF are achieved via *reification*, in which an RDF triple $p(S,O)$ is represented by, at minimum, four triples: $rdf:type(T,rdf:statement)$, $rdf:subject(T,S)$, $rdf:predicate(T,p)$ and $rdf:object(T,O)$ where T is a URI uniquely identifying the triple. Any other contextual information can be represented by triples in the form $p'(T,O')$ for any p' and O' . This is inefficient to implement, because it multiplies the number of triples that must be processed at least fivefold, and in some cases requires the generation and management of a globally unique ID for each triple.

Journaling. In an application, an RDF graph must be built procedurally, one triple at a time. In a distributed environment, a number of actors may modify a graph over time by adding and deleting triples. By logging each modification in a journal (e.g., log file), sufficient contextual information can be gathered to enable attribution and time-based decisions. For instance, the actor responsible for each modification, the type of modification (add/delete), and the time of the

modification can be recorded along with the triple being modified. Computing time and actor-scoped subgraphs from the journal is simple, but its worst-case performance is linear with the number of modifications. A journal is a convenient source of time and actor-scoped triples, but not an efficient means of accessing them by attribution and time.

n-Tuples. A simple way of adding information to a tuple is to add one or more terms. Declarative systems such as Prolog allow arbitrarily many terms per statement. RDF allows a fourth term for literal types, but is not generally extensible to n-tuples. Attribution and time can be added to RDF tuples by adding an actor term and a time interval term, or an actor term and two time terms for the start and end of the time interval. Since these n-tuples cannot be represented as RDF statements, a different representation is required which can be processed to produce time and actor-scoped subgraphs. Given such a representation, useful classes of problems concerning attribution and time can be resolved directly against it using abstractions that do not support efficient graph traversal, such as SQL.

4 Harvesting

Harvesting is a strategy for distributing data in which clients typically retrieve data from servers asynchronously and store it for later processing. Harvesting clients make decisions about which data to retrieve based on contextual information about how likely it is to have changed since it was last retrieved (e.g., HTTP caching directives) or by partially exposing their decision criteria to a server (e.g., OAI-PMH date range queries).

Harvesting Triples. It is not generally possible to harvest every triple from a triple store without contextual information, for the same reason that it is not generally possible to index every page on the web—both the web and RDF graphs are not guaranteed to be completely connected. Minimally, a graph store must expose information about which subgraphs may have changed over a given time interval to enable a harvesting client to walk the graph and find all information newer than the start of the time interval. Maximally, a graph store could expose a complete change log, as in the journaling strategy.

Multi-tier Approach. To improve efficiency, several processing stages can be interposed between actors modifying a triple store and reasoning engines carrying out high-level operations such as rules-based inference. Figure 1 introduces a three-tier approach.

In this example, an actor (“*joe*”) adds a triple $p(S, O)$ to a triple store. The triple is journaled along with the operation (“*add*”), actor (“*joe*”) and timing (t_k) information. An n-tuple store, consuming the journal entry, adds an entry recording an open time interval (t_k to t_∞). If the actor then deletes $p(S, O)$,

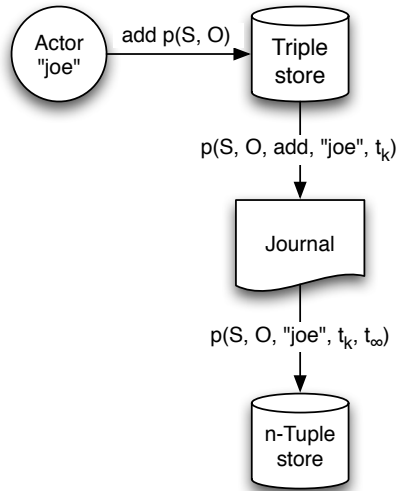


Fig. 1. Harvesting n-tuples from a journaling triple store

a journal entry $p(S, O, delete, "joe", t_{k+1})$ will be written and the n-tuple store will close the interval by modifying the relevant tuple as $p(S, O, "joe", t_k, t_{k+1})$.

The n-tuple store can make decisions about what triples to harvest using simple range and set membership queries. For instance it could trivially reject modifications from untrusted actors or delete all tuples with closed time intervals as a means of discarding non-current information.

Actors can write journal entries directly to a file or network stream instead of modifying a triple store. Any number of simple text formats suffice for representing and transporting journal entries, and journals generated by independent actors can be merged using simple, generic operations. The scenario is illustrated in Figure 2.

In this example, the triple store is populated with a triple added by means of an actor (“joe”) writing a journal entry. The advantage of this scenario is that because the journal is rolled up into an n-tuple store containing attribution and timing information about each triple, the triple store can be populated only with the triples relevant to some reasoning task with respect to attribution and timing.

5 A Practical Implementation

Harvesting triples can be accomplished through the multi-tier approach outlined above. In this section, a practical proposal for implementing that approach is outlined. The strategy outlined in this section is currently being used in NCSA’s Cyberenvironments project (<http://www.ncsa.uiuc.edu/Projects/index.html>) to link workflow provenance to social networking analysis codes.

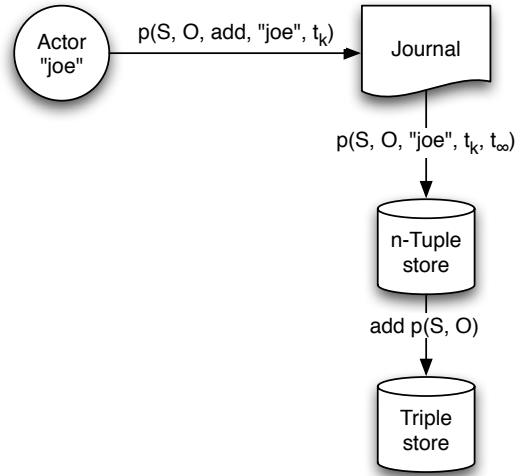


Fig. 2. Harvesting triples from a journaling actor

Journaling. In this section we present an RDF representation of journal entries and several format realizations, including standard RDF/XML and nonstandard extensions to N-Triples.

The proposed RDF representation is based on reification, which is used in conjunction with a proposed vocabulary representing attribution and timing. We propose that attribution and timing information for each triple be represented using Dublin Core `creator` and `date` properties, using an actor URI for the value of the `creator` element and an ISO 8601 timestamp for the value of the `date` element. To denote the operation the actor performed on the model (e.g., add or delete) we introduce the `wsw:operation` property where `wsw` is a prefix for the example “who said what when” namespace URI `http://tupeloproject.org/wsw`. Possible values for the `wsw:operation` property are `wsw:add` and `wsw:delete`.

```

<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:dc="http://purl.org/dc/elements/1.1/"
  xmlns:wsw="http://tupeloproject.org/wsw">
  <rdf:statement rdf:about="http://example.org/#genid927">
    <rdf:subject rdf:resource="http://example.org/#someSubject"/>
    <rdf:predicate rdf:resource="http://example.org/#somePredicate"/>
    <rdf:object rdf:resource="http://example.org/#someObject"/>
    <dc:creator rdf:resource="http://example.org/#joe"/>
    <wsw:operation rdf:resource="http://tupeloproject.org/wsw#delete"/>
    <dc:date rdf:dataType="http://www.w3.org/2001/XMLSchema#dateTime"
      2005-12-01T15:09:00Z
    </dc:date>
  </rdf:statement>
</rdf:RDF>

```

This syntax is verbose and requires the generation of a unique ID per journal entry. To mitigate these problems, we extend the N-Triples format so that each line contains not just the subject, predicate, and object but also the actor URI, operation, and an ISO 8601 timestamp. In this extended notation, the previous example is written on a single line as follows (\leftrightarrow denotes the continuation of a line):

```
<http://example.org/#someSubject>  $\leftrightarrow$ 
<http://example.org/#somePredicate>  $\leftrightarrow$ 
<http://example.org/#someObject>  $\leftrightarrow$ 
<http://example.org/#joe>  $\leftrightarrow$ 
<http://tupeloproject.org/wsw#delete>  $\leftrightarrow$ 
2005-12-01T15:09:00Z .
```

The extended N-Triples representation does not require generating a unique ID and can be compiled into standard representations should an application require it.

n-Tuples in SQL. This section outlines an SQL implementation of an n-tuple store which can be used to index a stream of journal entries. This example implementation has been designed for conceptual correctness and is not optimized.

The n-tuples can be represented using the following table definition:

```
CREATE TABLE ntuples (
  subject VARCHAR(255) NOT NULL,
  predicate VARCHAR(255) NOT NULL,
  object VARCHAR(255) NOT NULL, actor VARCHAR(255) NOT NULL,
  start_time TIMESTAMP DEFAULT CURRENT_TIMESTAMP NOT NULL,
  end_time TIMESTAMP,
  CHECK (end_time IS NULL OR start_time <= end_time),
  CHECK (NOT EXISTS
    (SELECT * FROM ntuples nt
     WHERE nt.subject = subject
           AND nt.predicate = predicate
           AND nt.object = object
           AND nt.actor = actor
           AND (start_time, end_time) OVERLAPS
              (nt.start_time, nt.end_time))));
```

The table corresponds closely to the abstract n-tuple model. In this representation, the `start_time` and `end_time` columns represent a half-open interval (containing the start time but not the end time) over which the triple is asserted to be a member of the set of all non-deleted assertions by the actor. A `NULL` `end_time` indicates that the actor has not deleted the triple since `start_time`. The `CHECK` constraints reject intervals whose end times precede start times, as well as overlapping identical statements by the same actor.

Adding a triple consists of performing an INSERT:

```
INSERT INTO ntuples
(subject, predicate, object, actor, start_time)
VALUES ('http://example.org/#someSubject',
       'http://example.org/#somePredicate',
       'http://example.org/#someObject',
       'http://example.org/#joe',
       '2005-11-28T03:09:00Z');
```

Deleting a triple consists of performing an UPDATE:

```
UPDATE ntuples
SET end_time = '2005-12-01T15:09:00Z'
WHERE subject = 'http://example.org/#someSubject'
AND predicate = 'http://example.org/#somePredicate'
AND object = 'http://example.org/#someObject'
AND actor = 'http://example.org/#joe'
AND end_time IS NULL;
```

Retrieving n-tuples based on trust is simple. Suppose the table `trusted_actors` contains a column called `actor` containing the URI of each trusted actor. The following query returns n-tuples created by trusted actors:

```
SELECT * FROM ntuples
WHERE actor IN (SELECT actor FROM trusted_actors);
```

Open Archives Implementation. The Open Archives Initiative Protocol for Metadata Harvesting (OAI-PMH) is a popular harvesting framework. This section outlines how the n-tuple abstraction can be adapted to OAI-PMH's model of a metadata collection.

OAI-PMH requires that a service expose one or more *sets of records*. A record is a time-stamped, identified item that can be retrieved in one or more *metadata formats*. It can also be deleted, and an OAI-PMH service is required to report that fact in response to queries about the record rather than acting as if the record never existed.

The OAI-PMH model matches the n-tuple model in several important ways. First, the n-tuple model contains information about the time at which triples were added and/or deleted. Second, sets of triples can be extracted from the n-tuple store and represented in several metadata formats, including RDF/XML and N3.

The most important *mismatch* between the n-tuple model and the OAI-PMH model is the granularity of description. In the n-tuple model, each triple is associated with attribution and timing information, but not otherwise identified. In OAI-PMH, a record is generally a *collection* of statements (e.g., a set of Dublin Core fields) associated with a single identifier and set of timing information. A closer match in the n-tuple model to OAI-PMH's concept of a record is the subject. The set of all non-deleted triples on a subject can be considered an

OAI-PMH record, whose identifier is the subject URI and whose timestamp is the most recent add or delete time from among the set of all triples, deleted or not, on the subject.

In SQL, the following query will retrieve all the non-deleted triples on a given subject (where the subject is `?s`):

```
SELECT subject, predicate, object FROM ntuples
WHERE subject = ?s AND end_time IS NULL;
```

This is based on the simplifying assumption that the n-tuple store does not allow an actor to delete a triple at some specific time in the future; that case could be handled by adding the clause `OR end_time > CURRENT_TIMESTAMP`.

Determining OAI-PMH timestamp for a record in SQL from the n-tuple table requires some temporal arithmetic, which can be accomplished with the following view:

```
CREATE VIEW oai_ts
AS SELECT subject, MAX(upd_time) upd_time
FROM
  (SELECT subject, MAX(start_time) upd_time FROM ntuples
   GROUP BY subject UNION
  SELECT subject,
   MAX(COALESCE(end_time, CURRENT_TIMESTAMP)) upd_time
   FROM ntuples GROUP BY subject)
GROUP BY subject;
```

Again, triples explicitly deleted in the future can be handled with a more complex query.

Actors in the n-tuple model make reasonable OAI-PMH sets, although a practical issue is how to map actor URI's to OAI-PMH set identifiers. A record's membership in a set corresponding to an actor can be determined by finding any n-tuple with the record's subject and the actor URI. The set of all sets corresponding to actors is also simple to compute in SQL from the n-tuple model using `SELECT DISTINCT(actor)`.

6 Conclusion

Harvesting heterogeneous information from multiple sources is critical to enabling collaborative e-science, and RDF provides a convenient abstraction for integrating heterogeneous information. To harvest RDF triples, it is useful to know “who said what when.” Implementing this second-order information using RDF reification scales poorly. Instead, extending the representation of RDF triples to include information about attribution and timing can enable harvesting decisions without the need for a fast triple store.

In this paper, I have outlined a three-tier approach to harvesting RDF triples in which a journal is “rolled up” into an n-tuple store before being compiled into

an RDF graph. The three tiers in the approach correspond to example implementations based on files, relational databases, and triple stores. The practical feasibility of the three-tier approach is demonstrated by harmonizing it with OAI-PMH, a standard protocol for metadata harvesting. The resulting approach supports reasoning in a dynamic, loosely-coupled, collaborative environment.

This strategy is currently in use as part of the CLEANER / CUAHSI Cyber-Collaboratory project at NCSA, a collaboration, data management, and workflow portal for environmental scientists and engineers. We use a logging API to capture user actions from workflow execution as well as asynchronous and synchronous collaboration and use the harvested triples to perform social network analysis and provide customized recommendations to users to help guide them through complex sets of resources and tools. For more information see <http://cleaner.ncsa.uiuc.edu/home/>.

References

1. Wong, S. C., Miles, S., Fang, W., Groth, P., and Moreau, L. *Provenance-based validation of e-science experiments*. In Proceedings of 4th International Semantic Web Conference (ISWC'05), volume 3729 of Lecture Notes in Computer Science, pages 801-815, Galway, Ireland, November 2005. Springer-Verlag.
2. Lagoze, C. and de Sompel, H. V. 2001. *The Open Archives Initiative: Building a low-barrier interoperability framework*. <http://www.cs.cornell.edu/lagoze/papers/oai-jcdl.pdf>. <http://citeseer.ist.psu.edu/lagoze01open.htm>
3. Heymans, S., Nieuwenborgh, D.V., Vermeir, D. *Preferential reasoning on a web of trust*. In Proceedings of 4th International Semantic Web Conference (ISWC'05), volume 3729 of Lecture Notes in Computer Science, Galway, Ireland, November 2005. Springer-Verlag.
4. Huang, Z., and Stuckenschmidt, H. *Reasoning with multi-version ontologies: a temporal logic approach*. In Proceedings of 4th International Semantic Web Conference (ISWC'05), volume 3729 of Lecture Notes in Computer Science, Galway, Ireland, November 2005. Springer-Verlag.
5. "Network Time Protocol," IETF RFC 958.
6. "RDF Semantics." W3C Recommendation, 10 February 2004. <http://www.w3.org/TR/rdf-mt/#Reif>

Mapping Physical Formats to Logical Models to Extract Data and Metadata: The Defuddle Parsing Engine

Tara D. Talbott¹, Karen L. Schuchardt¹, Eric G. Stephan¹, and James D. Myers²

¹ Pacific Northwest National Laboratory, P.O. Box 999 Richland, WA 99352, USA
{Tara.Talbott, Karen.Schuchardt, Eric.Stephan}@pnl.gov

² National Center for Supercomputing Applications, 1205 W. Clark St. MC-257
Urbana, IL 61801

jimmyers@ncsa.uiuc.edu

Abstract. Scientists, motivated by the desire for systems-level understanding of phenomena, increasingly need to share their results across multiple disciplines. Accomplishing this requires data to be annotated, contextualized, and readily searchable and translated into other formats. While these requirements can be addressed by custom programming or obviated by community standardization, neither approach has ‘solved’ the problem. In this paper, we describe a complementary approach – a general capability for articulating the format of arbitrary textual and binary data using a logical data model, expressed in XML-Schema, which can be used to provide annotation and context, extract metadata, and enable translation. This work is based on the draft specification for the Data Format Description Language and our open source “Defuddle” parser. We present an overview of the specification, detail the design of Defuddle, and discuss the benefits and challenges of this general approach to enabling discovery, sharing, and interpretation of diverse data sets.

1 Introduction

Scientists generate a wide range of data files in the course of their research. These files are generated from instruments performing measurements on physical systems, computer simulations predicting aspects of a physical system, and manually assimilated knowledge (often in spreadsheet form.) Individual file formats can vary greatly depending on the particular experimental requirements and often evolve rapidly over time. Motivated by the desire for systems-level understanding of complex phenomena, this data increasingly needs to be shared across disciplines and transformed for different analysis contexts. Beyond standard file formats, which have met with various levels of success [1-3], scientists employ strategies of custom programming and prescriptive parsers to support sharing and collaboration of their file data. Custom parsers can be effective and efficient but problems arise as the number of formats increases. Prescriptive parsers such as NetCDF [4] and HDF [5], where the data must adhere to a pre-specified, but self-describing format and structure have been successful within certain communities, but not taken hold in others. Where standards are successful, the standards tend to become legacy formats themselves over time as new methodologies or instruments are developed. Additionally, there is a

push to retain raw digital data for preservation purposes [6]. In short, non-standardized and legacy file formats will continue to play a crucial role in scientific research necessitating technologies to enable sharing, discovery and transformation of these formats.

The Extensible Markup Language (XML) allows us to represent the logical structure of data elements in a file, making it available to various tools, such as databases and query languages. XML tagged data can be easily manipulated using a higher level language such as XML Stylesheet Language Translation (XSLT) and formatted for viewing on multiple devices, or translated into different formats. However, most scientific data is not currently in XML and there are often benefits to maintaining custom formats. For example, XML tagged data tends to be quite verbose and not all data types, arrays in particular, are handled well. However, extending XML technology to handle arbitrary un-tagged, binary and textual files would make the extensive XML tools applicable to scientific data and provide analogous benefits.

Descriptive parsers can be used to link raw physical formats into a logical model expressed as XML. With this approach, the existing data structure, the format of the data types and the mechanisms to translate it are defined in a descriptor file. A generic parser engine ingests the descriptor and the data, applies the transformation, and produces the desired result. Such a generic engine can be applied to metadata extraction as well as data transformation to greatly reduce the effort required to discover and interpret legacy data, automate transfer of data from one program to another (e.g. acquisition to analysis to visualization), and support the reuse and fusion of data across multiple domains allowing scientific communities to discover, manage, and share diverse data sets while maintaining it in its original format.

2 Background

Recently, descriptive parser approaches have received increasing attention. One effort, the Binary Format Description (BFD) language, was based on the Extensible Scientific Interchange Language (XSIL) [7], a language designed for processing scientific data, including multiple streams and arrays. The BFD parser, in conjunction with XSLT, was used by scientific computing environments for the extraction of metadata and data translation. While successful in some cases, there were many cases where BFD capabilities were not rich enough. For example, BFD was unable to map to an arbitrary XML schema, requiring an additional XSLT translation. The research from the BFD effort contributed to the production of the parser described in this paper.

The BinX descriptive parser supports the description of the content, structure and physical layout (endian-ness, blocksize...) of binary files. BinX was designed to enable transparent transfer of data between diverse platforms. However, BinX was designed to support only binary files and, as with BFD, supports limited semantics [8]. An independent, but similar effort is the Earth Science Markup Language (ESML) which is built with the intent that users can write external files to describe the structure of any earth science dataset. Applications can utilize the ESML library to parse this description file and transparently decode the data format [9]. However, the library contains several limitations; not all features, such as handling multiple

wildcards, 'if' statements, or specific indexes of collections, are implemented, and, similar to BFD, a predefined XML model limits extensibility [10]. Another effort, designed primarily for understanding space data, the Enhanced Ada SubSet (EAST), allows users to describe a given data format and use tools to access data in that format [11]. Finally, the Universal Parsing Agent (UPA) was developed to ingest, transform, and add descriptive content markup to text data. UPA provides an accessible user interface and batch processing capabilities for handling large datasets [12]. All of these efforts have achieved success in their targeted communities but have limitations with respect to the type of data supported, extensibility, or expressiveness.

A recent development in descriptive parsers is the Data Format Description Language (DFDL) [13] specification from the Global Grid Forum. DFDL proposes to describe existing data formats, both binary and text, in a manner that makes the data accessible through generic mechanisms. DFDL is motivated by the realization that BFD, BinX, commercial tools, and domain specific efforts such as ESML, all shared a common goal and can use a common syntax while combining concepts of these languages. The specification is based on the XML Schema, which is used to define the structure and semantics of XML documents and to annotate schemas for the benefit of human readers and applications. In DFDL, XML's extensible annotation mechanism is used to describe the data and transformations needed to populate that logical model from the input stream. The input is a sequence of bytes and the output is an XML Information Model, i.e., a set of items from the XML Information Set [14]. The transformations may require several stages (e.g., from bytes to string, then from string to integer). The DFDL specification is still under development, but is expressive enough to handle many non-trivial parsing requirements.

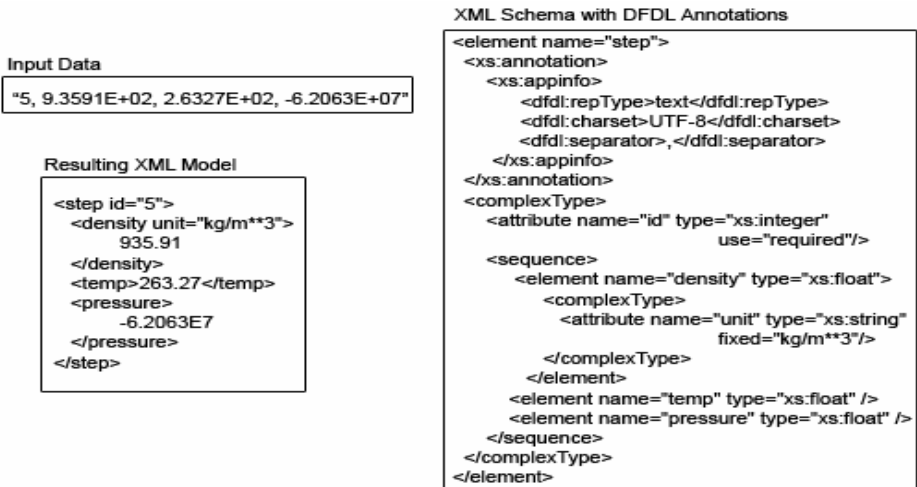


Fig. 1. Example of a DFDL schema with input data and results

For example, consider the UTF-encoded data, associated logical XML model, and sample DFDL schema shown in Figure 1. The DFDL schema is composed of an XML

Schema describing the XML model and DFDL specific annotations describing the format of the underlying data. As shown, a ‘*sequence*’ of element definitions describes the logical XML model that will appear in the result. The ‘*appinfo*’ annotations, known as properties, describe the format of the underlying data stream. The data type is defined as text by the ‘*repType*’ property. The ‘*charset*’ property defines how the incoming data stream is to be mapped to text. Finally, the ‘*separator*’ property describes how variable length text should be read. It can be defined as a regular expression or simple text string. This example shows a very limited subset of the available annotations and properties in order to provide a perspective on the DFDL approach. Table 1 lists important capabilities of DFDL. A more detailed description of DFDL capabilities is beyond the scope of this paper.

Table 1. Key DFDL Capabilities

Support for multiple streams	Conditional logic (if, choice, any)
Basic math operations (+, -, *, /)	Looping
Pattern matching for text/binary delimiters	External transforms
Reference values within schema (for sequence length, delimiters, etc.)	
Layering (hidden elements that can be referenced, but do not display in output)	
Extensibility of basic capabilities of the DFDL parser to allow custom types and conversions	

In order to help define the components necessary in the language, several DFDL parsers are currently in development. One such implementation is the open-source Defuddle parser [15]. Defuddle supports translation and metadata extraction of arbitrary text and binary data through the use of DFDL schema descriptor files. It also optionally supports the application of style sheets to the output. The Defuddle parser is both a proof of concept of the DFDL specification and a mechanism for testing concepts which can feed back into the specification process. A specific aim of Defuddle is to demonstrate that an efficient, generic parser can be built and that such a parser can effectively address real-world examples.

3 Parser Design

Our design leverages existing tools for automatically parsing XML documents within the context of a logical model. Providing a layer of automation that makes it easy to manipulate XML encoded data in terms of a higher level logical model rather than dealing with the low level node structure directly [16]. As a result, we chose to extend a Java/XML binding compiler based on the Java Architecture for XML Binding (JAXB) specification. JAXB provides a convenient way to use XML Schema to automatically parse XML instance documents into Java classes corresponding to that schema. From a design standpoint, this solution provides an off-the-shelf ingestion engine for the logical model (XML Schema), a dynamic logical model compiler (Java classes), and an XML document generator for streaming data from the classes to XML.

Figure 2 illustrates the conceptual design of the Defuddle parser. At run-time, the schema is ingested and processed to generate Java classes representing components of the logical model. These classes are then compiled using the standard Java compiler. The translation of the input data source(s) is then initiated using the JAXB XML marshaller. As the java objects are streamed by the parser, the logical model is formed by loading the required values from the data file. The XML model is streamed to an output that can then be processed by standard XML tools. The class generation process is performed automatically before translations are performed, but the compiled classes can be cached to improve performance on subsequent runs.

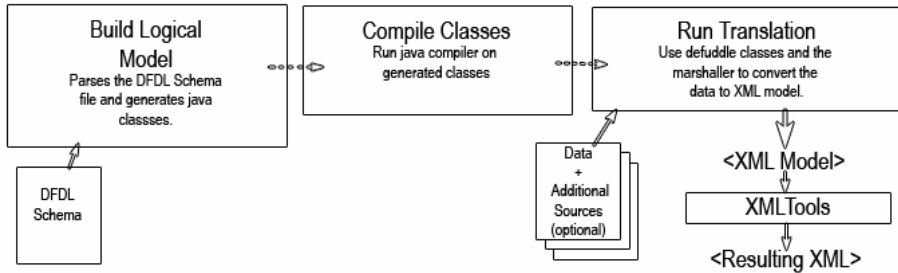


Fig. 2. Conceptual design of the Defuddle Parser

Defuddle is based on the Apache JaxMe project [17], an implementation of the JAXB specification. Defuddle extends the JaxMe class generator to provide the functionality needed to load data into the model and complete the transformation. Leveraging JaxMe greatly simplified the development of Defuddle. Figure 3 illustrates the types of classes generated by JaxMe and the Defuddle extensions that implement various features. Each complex type is represented by three classes: the type implementation, type driver, and type handler. Within the type implementation, values such as elements and attributes are accessed from the data stream using `get<Name>` methods. Vanilla JaxMe generated type implementations store and return the values, Defuddle adds content to these ‘get’ methods, which uses the annotation handlers and data provider along with other built-in Defuddle classes, such as the type readers, condition evaluators, and external transforms, to parse the required data.

While the complexType implementation classes provide information to parse individual values, the parser needs additional information to understand the structure of the data. This includes the order of the elements, the location of sequences, and the type of data to be marshaled. This functionality is found in the complexType drivers. The basic ordering and ‘get’ calls are generated by JaxMe. Defuddle extensions check for layers, hidden elements, and control sequences of unknown lengths. They also choose the correct element in conditional statements, and pass in the data provider and annotation values. The third kind of generated class is the complexType Handler; these classes are generated almost entirely by the JaxMe generator and chiefly control the marshalling of the classes to XML, ensuring the correct state when starting/closing elements and writing data.

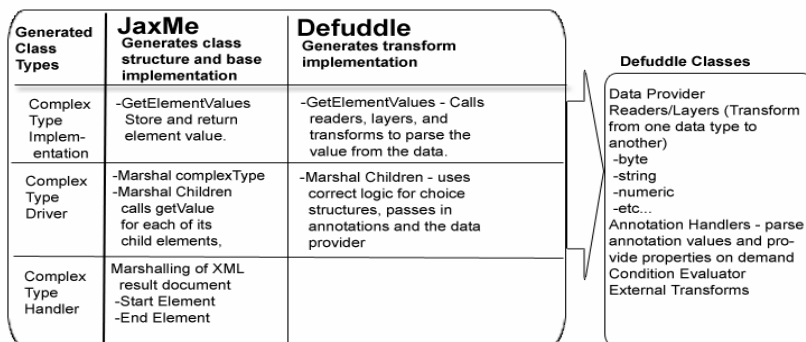


Fig. 3. Defuddle Extensions to JaxMe

Along with the JaxMe extensions, Defuddle contains additional classes to aid in parsing as shown in the right side of Figure 3. Various readers are used for converting values from one type to another, for example from byte to string, from string to multiple strings, and from string to int. Each of these readers use the annotation properties specified in the schema. Defuddle also uses a data provider to retrieve from the data stream, referencing other values in the schema, or for handling multiple input sources. When evaluating conditions, the annotations are passed on to a condition evaluator to determine the correct element to read. Defuddle also supports the calling of external transformations and integrating the results into the Defuddle transformation.

4 Discussion

Defuddle currently supports all of the features listed in Table 1. To validate the accuracy of the parser, a collection of example schemas and files have been composed. These cover a broad range of capabilities such as reading basic binary/ascii numbers, basic math operations, the ability to reference other elements within the schema, and the ability to read from multiple files or input streams. In practice, it is necessary to use a combination of these features. We have demonstrated the parser capabilities on several types of actual formats from the biological and chemical sciences including: CHEMKIN binary solution files, NWChem Molecular Dynamics property files and unstructured output files, and MicroArray and Protein-Protein Interaction Spreadsheets.

One goal of Defuddle is to demonstrate that an efficient, generic parser can be built to address real-world examples. Performance is of particular interest; can a generic descriptive parser perform as efficiently as a custom parser? A generic, pre-compiled, parser can be optimized based on the types of data and access pattern to make use of lazy parsing, avoiding unnecessary reading and caching of data. Pre-compiled schemas can also be cached to eliminate the code generation /compilation cycle. We are researching ways to better enable rapid, random access to partial data sets from tightly structured data, and to support the parsing of large data sets (through memory

mapping and streaming). Additionally, with a code-generating approach, the binder can make choices when creating the schema classes to handle much of the actual parsing, such as choosing the type and length of data to be read, pre-computing the size of each data element, and requesting individual elements in tightly structured data can be served by seeking to the exact point in the data stream rather than parsing all intermediate values. For example, for a list of 100 binary floating point numbers, the location of the x th value can be computed based on the actual size of the numbers and the index of the desired number. Unfortunately this enhancement is not possible when reading varying length text, in which one must look for a separator between each element, and the size of the elements can vary based on the data being processed. Even with variable length data, intelligent parsing can still be achieved by estimating the length of the next element to read before evaluating for a delimiter, based on the size of previous elements in the sequence. If the delimiter is found before the end of the text read, the parser is able to backtrack to the position in the stream immediately after the delimiter, ensuring that no data is accidentally skipped. This estimation is often very close and speeds up parsing considerably.

When retrieving data, a more complex transformation than mere extraction may be required; this can be handled through the idea of layering. With layering, the user can describe intermediate forms of the data which are not represented in the final result. These layers are represented in the DFDL language through the use of XML Schema annotations which specify how and where each layer should be read in a stream; they also specify a name which can later be used to reference data within the layer. A layer can be accessed using annotations similar to the method used to reference other elements within the schema.

Put into practice, this type of generic parsing capability can provide a cornerstone to data sharing and collaboration environments by providing metadata extraction, translations, data slicing, and data fusion capabilities. For example, the Scientific Annotation Middleware (SAM) project provides configurable, automated metadata extraction of uploaded resources [18]. Combinations of XSLT stylesheets, Defuddle schemas, and web services are registered with SAM and run dynamically to extract metadata. For example, when a binary data file is uploaded to SAM, registered DFDL and XSLT files are accessed to generate relevant properties and store them as metadata, allowing users to automatically capture annotations. A similar mechanism can be used to provide data views - for example dynamically generated HTML pages or pages invoking Java applets for a browser-based view of the data [19]. Combined with a user environment such as the Collaboratory for Multi-scale Chemical Sciences (CMCS) [20], users can contribute data that is readily available for other users to browse, search, and access in a format suitable for their use. The availability of Defuddle is expected to reduce the number of custom translators, serve as a library of translations within applications, and provide the querying of subsets from large files. If the data description and subset queries were associated with persistent identifiers such as Life Science Identifiers (LSIDs), it should be possible to create virtual persistent identifiers for substructures and to resolve and retrieve substructures on demand [21].

5 Conclusion

In the paper, we presented a general “descriptive parser” approach to mapping physical formats to logical XML representations. This approach, based on the Data Format Description Language specification, uses data descriptions based on XML Schema extensions. Once in XML, off-the-shelf XML solutions can be applied to readily transform data, extract data and metadata, or to query the data. We detailed the design and implementation of an open-source parser engine known as Defuddle. Using real-world file formats from the chemical and biological sciences, we demonstrated that the current capabilities defined in DFDL and implemented in Defuddle are already capable of parsing a diverse set of formats. While the DFDL specification is still a work in progress, Defuddle has proved to be a useful tool in guiding specification activities and is being used to explore how extensibility can be integrated with the basic feature set.

In the future, our research will focus on extensions for internal and external transforms, layering transformations, and optimally generating data subsets from XSL translation and XPath queries. The latter feature will require smart parsing and the predetermination of the position of elements within data streams. Such features, together with the already existing capabilities, enable a range of light-weight, loosely-coupled data integration and data virtualization systems needed to support multi-disciplinary research on complex phenomena.

Acknowledgment

The research described in this paper was conducted under the Laboratory Directed Research and Development Program at the Pacific Northwest National Laboratory, a multiprogram national laboratory operated by Battelle for the U.S. Department of Energy under Contract DE-AC05-76RL0 1830. The authors acknowledge Robert McGrath and his work on a DFDL primer as well as helpful discussions and ongoing collaborations with members of the DFDL working group.

References

1. Critchlow, T., and Lacroix, Z., eds., *Bioinformatics:Managing Scientific Data*. July 2003. Morgan Kaufmann.
2. Lancashire, R., Davies, T., *Spectroscopic Data: The Quest for a Universal Format*, Chemistry International, Vol. 28 No. 1, January-February 2006
3. Robins, K.D., “Formatting Standards”, http://www.ofcm.gov/sai/proceedings/pdf/02_panel2-3.pdf
4. netCDF Unidata: “netCDF”: <http://my.unidata.ucar.edu/content/software/netcdf/index.htm>
5. HDF: <http://hdf.nsa.uiuc.edu/>
6. Davies, T., “Cometh a Digital Dark Age?”, *Chemistry International* Vol 24, No. 6, November 2002
7. Extensible Scientific Interchange Language: <http://www.cacr.caltech.edu/SDA/xsil/>
8. Binary XML Description Language: <http://www.edikt.org/binx>
9. Environmental Science Markup Language: <http://esml.itsc.uah.edu/index.jsp>

10. Environmental Science Markup Language: <http://esml.itsc.uah.edu/limitations.html>
11. Enhanced Ada Subset (EAST): <http://east.cnes.fr/english/index.html>
12. Whiting MA, WE Cowley, NO Cramer, AG Gibson, RE Hohimer, RT Scott, and SC Tratz. 2005. "Enabling Massive Scale Document Transformation for the Semantic Web: the Universal Parsing Agent." Proceedings of the 2005 ACM symposium on Document Engineering. pp 23-25. ACM Press, New York, NY
13. Data Format Description Language: <http://forge.gridforum.org/projects/dfdl-wg>
14. John Cowan and Richard Tobin (eds), "XML Information Set" W3C Working Draft 16 March 2001, <http://www.w3.org/TR/xml-infoset> .
15. Defuddle Sourceforge Project: <http://sourceforge.net/projects/defuddle>
16. Java Architecture for XML Binding : <http://java.sun.com/webservices/jaxb>
17. Apache JaxMe: <http://ws.apache.org/jaxme/>
18. Scientific Annotation Middleware: <http://collaboratory.emsl.pnl.gov/sam/>
19. Talbott TD, MR Peterson, J Schwidder, and JD Myers. 2005. "Adapting the Electronic Laboratory Notebook for the Semantic Era." 2005 International Symposium on Collaborative Technologies and Systems, pp. 136-143. IEEE Computer Soc., Los Alamitos, CA.
20. Collaboratory for Multi-Scale Chemical Science: <http://cmcs.org>
21. Myers, J. "Fine-grained References into Binary Data and Data Virtualization Services", Presented at W3C Workshop on Semantic Web for Life Sciences 27-28 October 2004, Cambridge, Massachusetts USA

Annotation and Provenance Tracking in Semantic Web Photo Libraries

Christian Halaschek-Wiener, Jennifer Golbeck, Andrew Schain, Michael Grove,
Bijan Parsia, and Jim Hendler

University of Maryland, MIND Lab, 8400 Baltimore Ave., College Park, MD 20742, USA
{halasche, golbeck, hendler}@cs.umd.edu, andrew.schain@nasa.gov
mhgrove@hotmail.com, bparsia@isr.umd.edu

Abstract. As the volume of digital images available on the Web continues to increase, there is a clear need for more advanced techniques for their effective retrieval and management. In this paper, we present a domain independent framework for both annotating and managing images on the Semantic Web. We introduce a tool that facilitates creating and publishing OWL annotations of image content to the Semantic Web. This is loosely coupled with a Semantic Web portal with provenance tracking. We illustrate the effectiveness of this system with an implementation of the approach and describe a hypothetical use case that resulted in a proof-of-concept designed in collaboration with NASA.

1 Introduction

As the scale and infrastructure of the Internet have dramatically increased over the past years, we have seen the incorporation of various digital media types onto the Web, including images, video, and audio. As production of digital media content continues to grow in the commercial and home use markets, and as Internet access and wider bandwidth become even more pervasive, we can anticipate a continued increase of these complex (non-textual multimedia) data types being made available on the Web. Due to the format of such media, standard indexing techniques commonly used on text-based Web content, such as keyword-based approaches [2], are of little use. Given the volume of unstructured digital media, it is clear additional approaches and techniques must be developed to allow for their effective management and accurate retrieval.

Over the past few years, various approaches have been proposed to effectively retrieve and manage digital image content on the Web. Traditionally, these have included techniques such as building keyword indices based on image content [7, 9], embedding keyword-based labels into images [7], analyzing text immediately surrounding images on Web pages [4], etc. More recently, there has been a research focus to develop techniques to annotate the content of images on the Semantic Web, using languages such as RDFS and OWL [1,3,5,6,11].

Recent efforts have largely focused on mapping low-level features of images to ontological concepts [1,3,11] and have involved the development of tools that are closely tied to domain specific ontologies for annotation purposes [6,8]. Additionally,

past approaches have largely left unaddressed image metadata management and advanced interaction (browsing and search capabilities) that is enabled by employing Semantic Web technologies. While substantial progress has been made, we see the need for further work in defining a more generic approach for annotating and managing digital images on the Web.

In this work, we present an approach that provides generic, domain independent flexibility for publishing annotations of digital image content to the Semantic Web, as well as a mechanism for managing such annotations and tracking their provenance through a highly customizable, ontology-backed Semantic Web portal. Through the loose coupling of the annotation and management components of our approach, a seamless environment is provided in which users can annotate, share, and manage their digital images on the Semantic Web.

2 Motivation and Approach Overview

To understand the generic requirements that have driven the approach presented here, a representative use case based on a collaboration with NASA. While this motivation is presented in the context of NASA, we feel the model is sufficiently generic, thus capturing the general issues associated with managing metadata of digital images.

As an enterprise, NASA has hundreds of thousands of images, stored in different formats and locations, at different levels of availability and resolution, and with associated descriptive information at various levels of detail and formality. NASA also generates thousands of images on an ongoing basis that are collected and cataloged, often in accordance with needs of the image creator's specific disciplines and domain (preliminary investigators, mission specialists, public affairs, etc.). It is clear that a mechanism is needed to catalog all the different types of image content across different domains. Information is required about both the image itself (creation date, dpi, source, etc.) and also about the content of the picture (contains a satellite, astronaut, etc). The associated metadata must be maintainable and extensible so associated relationships between images and data can evolve cumulatively within a discipline or branching into other disciplines. The service must be available to a global consumer population but should be flexible enough to enforce restriction based on content type, ownership, authorization, or time.

A promising strategy for such image management requirements is an annotation environment that enables both providers and users to annotate information about images or regions in images using concepts in ontologies (OWL and/or RDFS). Thus, subject matter experts and consumers (regardless of their location) will be able to assert metadata elements about images and publish their annotations to the Semantic Web. There, such digital image annotations can be harvested and merged, resulting in advanced browsing, searching, and management.

We generalize these (NASA specific) high level requirements into the following application independent requirements: support for *ad hoc* ontology-based annotation of images on the Web, enabling support for annotation with respect to *any* domain; the ability to make assertions about images and the contents of specific regions in images; the ability to automatically publish annotations to the Semantic Web, where they can be shared, indexed, and maintained; provide a metadata management facility

for interacting with and maintaining image metadata that is accessible to a global community – the Semantic Web; the ability to accumulate metadata about a specific image over a period of time from different sources.

Given these requirements, we present a loosely coupled approach that provides generic, domain independent flexibility for creating and publishing annotations of digital image content to the Semantic Web, as well as a mechanism for managing such annotations through a highly customizable, ontology-backed Semantic Web portal.

3 Implementation Details

The first component of the approach presented in this work is a digital image annotation environment. In this section we present PhotoStuff, a semantic annotation environment.

3.1 Digital Image Annotation – PhotoStuff

PhotoStuff is a platform independent, open source, image annotation tool that allows users to annotate an image and its regions with respect to concepts from any number of ontologies specified in RDFS or OWL. PhotoStuff provides functionality to import images, their embedded metadata, ontologies, and instance-bases. In the tool, users can perform markup, and export the resulting annotations. The tool provides users the ability to load multiple OWL and/or RDFS ontologies, allowing annotation of image content with respect to any concept, defined in any number of ontologies. The ability to annotate images with respect to any ontology is extremely important because the content of images can span multiple domains; frequently, a single ontology cannot capture the complexity of the content.

In PhotoStuff, an ontology-based approach has also been adopted in order to make statements regarding the high level concepts depicted in images. An ontology is used to provide the expressiveness required to assert what is depicted within an image, as well information about the image itself. In this work, an image-region ontology¹ has been specified, using OWL, which defines a set of concepts and their relations for images, videos, regions, and depictions.

To demonstrate the use of PhotoStuff, Figure 1 shows a screenshot of the tool in which a user is marking up information about an astronaut taking a space walk. The ontologies are visualized in both a class tree and list, depicted in the far left pane of the tool. In this example, the FOAF (Friend of a Friend) ontology has been loaded, as well as a Shuttle Crew ontology that is expanded in the window. This allows the user to choose concepts from both ontologies to mark up the photograph and its sub-regions.

In this approach, users can annotate the entire image, or selected regions. Users highlight regions around portions of images loaded in PhotoStuff. Figure 1 illustrates this with a region drawn around the astronaut. Classes can be dragged onto the image or into any region creating a new instance of the selected class. When a class is used, a form is dynamically generated from the properties of the selected class. With region

¹ Image-Region Ontology: <http://www.mindswap.org/2005/owl/digital-media>

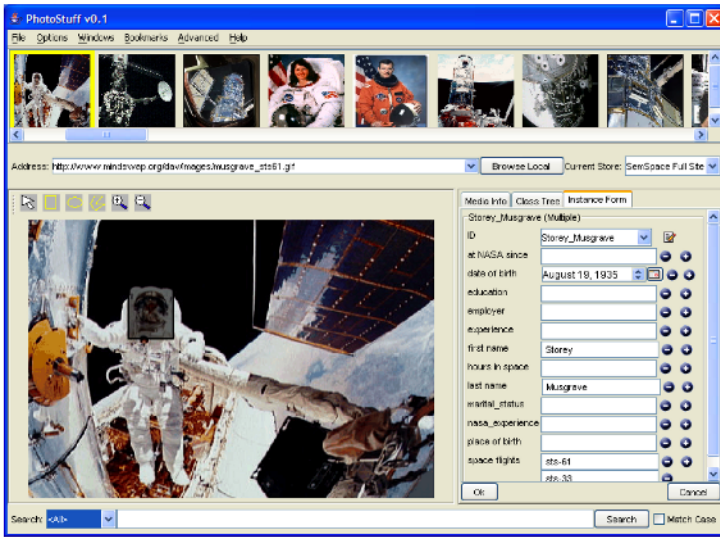


Fig. 1. PhotoStuff Screenshot

support, metadata can be more closely tied to the depiction it describes. Instead of simply stating that a photograph depicts several people, the metadata will contain coordinates for the regions of the photo that contain the depictions. The region is also semantically linked to the image, maintaining the connection between the image and the instance.

Additionally, the approach presented here leverages current efforts in multimedia format standardizations that provide support to embed image metadata in actual image files. For example, the JPEG file format provides support for embedding a standard set of markers in the file header, defining metadata elements including file size, width/height, pixel density, etc. The Exchangeable Image File Format (EXIF), also provides metadata in the form of camera specific information (camera make, model, orientation, etc.).² Our approach takes advantage of this existing metadata by extracting and encoding it into RDF/XML so it is accessible on the Semantic Web.

As mentioned earlier, PhotoStuff, and the approach in general, maintains a loose coupling with a Semantic Web portal. As briefly discussed before, there are three ways in which PhotoStuff interacts with the portal: retrieving all instances that have been submitted to the portal, submitting generated RDF/XML, and uploading local images so they can be referenced by a URI (thus allowing them to be referenced using RDF/XML). The following section outlines the metadata management and browsing functionality provided through the loose coupling of the annotation environment with the Semantic Web portal.

3.2 Image Metadata Management

Upon the completion of image annotation, the approach provides the capability for publishing resulting markup to the Semantic Web. This is accomplished through the

² EXIF Homepage: <http://www.exif.org/>



Fig. 2. Instance Depictions and Co-Region Browsing

coupling of the annotation environment with an ontology-backed, Semantic Web portal. Our existing work on a Web portal based on Semantic Web technologies (OWL) has been extended to provided communication with PhotoStuff. It is noted here that the Web portal’s functionality extends what is presented here and is an ongoing project within the MINDSWAP³ research group. Details are provided here through one of its configurations in the context of a proof-of-concept, SemSpace⁴, developed as an experiment with NASA. A variety of other domain configurations have been developed at MINDSWAP; all configurations provide the same functionality, only differing by the ontologies and instances maintained by the system.

The portal technology is flexible enough to be used in a variety of domains, as it is not limited in the number of ontologies that it can manage; thus for the purpose of this work, any ontology can be used to annotate an image. The portal is designed to use information from the various ontologies to guide the display of and interaction with metadata and the site in general. The main interface for browsing images is driven by the underlying class of each instance, thus providing a high level view of all the metadata of images that have been annotated using PhotoStuff.

The portal provides the ability to browse data associated with instances, images, image regions, and to search metadata in the collection (in Figure 2). The portal component also provides various management capabilities. Metadata submissions can be audited, edited, or removed. Provenance information (submitter name, email, etc.) from all submissions is maintained and editable. For each statement, the provenance information is also provided when the user hovers their mouse over any annotation.

³ MINDSWAP Research Group: <http://www.mindswap.org/>
⁴ SemSpace Homepage: <http://semSPACE.mindswap.org/>

3.2 Provenance Management

As mentioned previously, PhotoStuff and the portal technology exploit and provide support for management of provenance data. When users submit annotations to a portal, their user name and a comment is required for input. When the portal receives this data, a timestamp is additionally recorded. This provenance data is stored, along with the annotations in a RDF triple store, RDFLib⁵.

Each time a new submission is received, a new submission resource is created and stored in the triple store. Additionally, the submission object has various attributes related to it, including the submitter, timestamp, etc.

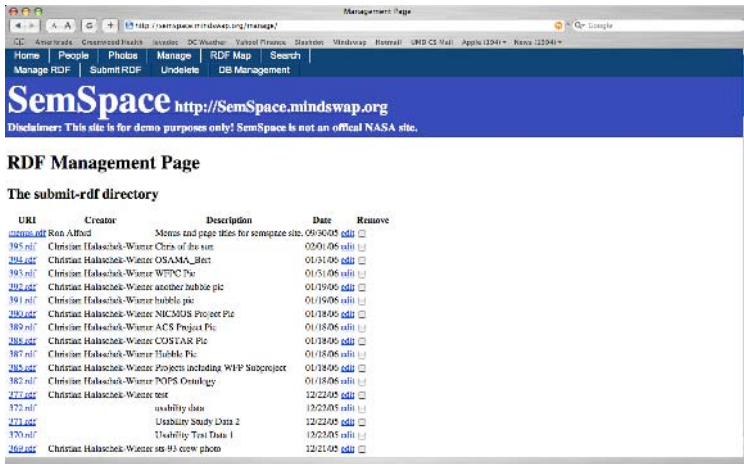


Fig. 3. RDF Submission Management Interface with Provenance Data



Fig. 4. An excerpt of a page in the site showing Mouse Pop-ups of provenance data when site browsing

⁵ RDFLib Project Homepage: <http://rdflib.net/>

The portal technology provides a submission metadata management page, in which submission can be edited and/or removed. In order to assist system administrators in finding submissions, the provenance data is presented in the interface (see Figure 3).

Provenance data maintained on the portal is also utilized to enhance the user experience when browsing data on the portal. When resources are browsed on the website, pop-ups are provided if provenance data is known about the particular data be viewed. This is shown in Figure 4.

4 Discussion and Future Directions

The approach discussed in this paper allows ad hoc, manual annotation of image content. This provides a cumulative technique where metadata can be incrementally added or repurposed for future users on a per-need basis. While manual annotation is essential for such ad hoc additions or edits, it can prove to be quite time consuming. This may be slightly alleviated through use of various image processing and automated vision techniques. First, region segmentation techniques may be used to suggest possible regions of interest. Additionally, image-processing techniques could potentially be used to recognize similar regions among photos, allowing the tool to suggest potential instances that may be depicted in the image. By exploiting similarity in images that are part of the same "album", automated recognition techniques can be used to take a first pass at labeling parts of the image. Once images are automatically labeled, users can then simply verify the resulting annotations [10].

Additionally, in this work it has been observed that generating effective, yet generic forms based on class definitions can be quite difficult (in this context, instance creation forms are generated when classes are dragged into image). We have adopted an approach in which the form is directly built from the underlying properties of the class. While this approach is a plausible first step, it can result in a very messy or congested form. We would like to explore allowing the user to create custom forms for classes. Additionally, we would like to investigate allowing ontology creators to embed HTML forms or XForms⁶ into comments on class definitions.

5 Conclusions

In this work we have presented a generic, domain independent framework for annotating and managing digital image content using Semantic Web technologies. We loosely couple an annotation component with a Semantic Web portal that supports browsing, searching and managing digital image annotations and provenance information. Additionally, we have provided details of an open source implementation of this framework and an overview of a representative proof-of-concept. Potential future work includes automating portions of the annotations process, possibly by using image processing and computer vision techniques. We also plan to extend our work here to support annotation of additional digital media types, including video and audio.

This work was supported in part by grants from Fujitsu, Lockheed Martin, NTT Corp., Kevric Corp., SAIC, the National Science Foundation, the National

⁶ XForms 1.0: <http://www.w3.org/TR/xforms/>

Geospatial-Intelligence Agency, DARPA, US Army Research Laboratory, and NIST. We would like to thank NASA for their help in documenting requirements for this effort. We would also like to thank Daniel Krech, Ron Alford, Amy Alford, Grecia C. Lapizco-Encinas, and Aditya Kalyanpur for all of their contributions to this work.

References

1. Addis, M., Boniface, M., Goodall, S., Grimwood, P., Kim, S., Lewis, P., Martinez, K. and Stevenson, A. SCULPTEUR: Towards a New Paradigm for Multimedia Museum Information Handling. Second International Semantic Web Conference (2003) 582 -596
2. Brin, S., and Page, L. The Anatomy of a Large Scale Hypertextual Web Search Engine, *In the Proceedings of the 7th International World Wide Web Conference* (1998)
3. Dupplaw, D., Dasmahapatra, S., Hu, B., Lewis, P., and Shadbolt, N. Multimedia Distributed Knowledge Management in MIAKT. *ISWC 2004 Workshop on Knowledge Markup and Semantic Annotation*. Hiroshima, Japan, November 2004
4. Frankel, C., Swain, M., and Athitsos, V. Webseer: An Image Search Engine for the World Wide Web, Tech. Report TR-96-14, Computer Science Dept., Univ. of Chicago, July (1996)
5. Hollink, L., Schreiber, G., Wielemaker J., and Wielinga. B. Semantic Annotation of Image Collections. Knowledge Capture - Knowledge Markup & Semantic Annotation Workshop (2003)
6. Lafon, Y., and Bos, B. Describing and Retrieving Photos Using RDF and HTTP. W3C Note available at: <http://www.w3.org/TR/photo-rdf/> (2002)
7. Rui, Y., Huang, T. S., and Chang, S. F. Image Retrieval: Current Techniques, Promising Directions, and Open Issues. *Journal of Visual Communication and Image Representation*, Volume 10 (1999), pp. 39-62
8. Schreiber, G., Dubbeldam, B., Wielemaker, J., and Wielinga, B. Ontology-Based Photo Annotation. *IEEE Intelligent Systems*, 16(3) (2001) 66-74.
9. Smith, J. R., and Chang, S. F. An Image and Video Search Engine for the World Wide Web. *Proc. SPIE 2670 Storage and Retrieval for Still Image and Video Databases IV*, SPIE, Bellingham, Wash., (1996) pp. 84-95.
10. Suh, B., and Bederson, B. Semi-Automatic Image Annotation. University of Maryland Computer Science Department Technical Report, HCIL-2004-15, CS-TR-46 (2004)
11. Bloehdorn, S., Petridis, K., Saathoff, C., Simou, N., Tzouvaras, V., Avrithis, Y., Handschuh, S., Kompatsiaris, I., Staab, S., and Strintzis, M. G.: "Semantic Annotation of Images and Videos for Multimedia Analysis", 2nd European Semantic Web Conference, 2005.

Metadata Catalogs with Semantic Representations

Yolanda Gil, Varun Ratnakar, and Ewa Deelman

USC Information Sciences Institute
4676 Admiralty Way
Marina del Rey, CA 90292
gil@isi.edu, varunr@isi.edu, deelman@isi.edu

Abstract. Metadata catalogs store descriptive information about logical data items. These catalogs can then be queried to retrieve the particular logical data item that matches the criteria. However, the query has to be formulated in terms of the metadata attributes defined for the catalog. Our work explores the concept of virtual metadata, where catalogs can be queried using metadata attributes not originally defined in the catalog. We use semantic web standards, where new metadata attributes can be taken from shared ontologies and can include expressive axioms to define the new terms. We have implemented a virtual metadata catalog as an extension of the Metadata Catalog Service (MCS), using the Web Ontology Language (OWL) and a reasoning engine to map queries of temporal nature in several metadata catalogs.

1 Introduction

An integral part of today's large-scale science is the identification and access of large data sets. To support a scalable solution, many systems distinguish between data cataloging and data storage. Data cataloging is designed for ease of publication of data characteristics (metadata attributes including provenance information) and for ease of querying for data products based on the desired metadata attributes. Having uniquely identified the desired data products (by obtaining an identifier) then enables data access from an appropriate storage location. Metadata attributes and unique identifiers are stored in metadata catalogs, often accessible as services [3,8].

The process of data discovery by querying data sources may be quite complex, because there may be several heterogeneous metadata catalogs that are being published by a community. For example, in the astronomy community and under the umbrella of the National Virtual Observatory project (NVO, www.us-vo.org), astronomers publish data collected by a variety of sky surveys taken from the both ground and space-based telescopes. The surveys span a whole range of spectra from gamma- and X-rays, optical, infrared, through to radio. Each catalog may contain tens of millions of objects. Because the catalogs came online in different period of time, were published by different organizations and deal with different surveys, the metadata attributes are not common across the catalogs. Obviously these differences make it very hard to easily discover desired data products. The problem is that the metadata attributes are not defined or related to one another according to their meaning. Clients must figure out manually the meaning of the attributes, identify

what are the relevant ones to query, and formulate queries that include all possibly relevant metadata attributes resulting in redundancies in the query expression. This poses limitations in terms of the practical usability of these catalogs as well as the potential of existing approaches to scale up to larger and heterogeneous collections of data sources. These problems arise in similar projects in other disciplines, such as the Grid Physics Network (the GriPhyN project, www.griphyn.org) and the Southern California Earthquake Center's Community Modeling Environment (the SCEC-CME project, www.scec.org). There, a central goal is the distributed management of data collections that evolve over time and the consumption of those collections by an entire community with very diverse uses and possibly conceptualizations of the data.

This paper describes an approach that augments the existing metadata catalogs with semantic representations to create *virtual metadata catalogs*. Virtual metadata attributes are mapped to the original attributes that appear in the metadata catalog. This is analogous to the concept of virtual data in GriPhyN [1] or a virtual observatory in NVO [2], where a system can generate the data requested based on its description whether it already exists or it has to be generated from data that already exists. The definitions of the virtual attributes are represented declaratively, as well as any constraints that represent how attributes are interrelated. A query formulated in terms of the virtual metadata can be automatically expanded and translated into the original metadata attributes using the virtual metadata definitions and mappings. This can be done by using a logic reasoner that can handle expressive representations of definitions and relations. With this approach, integrating metadata catalogs can be done through shared ontologies and standard terminologies, decoupling the query formulation from the virtual metadata handling and from the particular metadata attributes that appear in the catalog.

In prior work we developed Artemis [3], a query mediator for metadata catalogs that used semantic representations to integrate several metadata catalogs. Artemis uses a centralized approach with a single reasoner that incorporates all the representations and mappings to all the metadata catalogs. The approach we take in this paper is decentralized in that a reasoner is associated with each metadata catalog. We describe our implementation of a virtual metadata query handler that uses semantic web technologies such as the Web Ontology Language (OWL) standard [4] to support virtual metadata queries for the Metadata Catalog Service (MCS) [5]. MCS is a metadata catalog we previously developed to support the publication and query operations on a variety of scientific metadata. It provides an extensible schema and an API that enables easy query and publication capabilities. In future work, we plan to combine this virtual metadata query handler with the query mediator used in Artemis to support the integration of distributed metadata catalogs in a decentralized manner.

The paper begins showing how semantic representations can express declaratively the meaning of metadata attributes, so that automated reasoners can derive relations and infer connections among attributes and data sets. Next it describes briefly existing standards for semantic representations, including RDF schemas and OWL. With this background in hand, the paper then introduces our approach to create virtual metadata attributes and catalog services. The paper ends with a description of an implemented virtual metadata catalog service and discusses important future work.

2 The Need for Semantic Representations of Metadata Attributes

Unless the meaning and interdependencies of metadata attributes are represented declaratively with expressive languages, metadata catalogs have limitations in the way they can be used to query data. An important limitation arises when the meaning of the attributes is not represented explicitly and is instead implicit in the name of the metadata attribute. For example, an attribute named `execution-time` could mean elapsed time or CPU time. The right answer can only be determined manually by looking up the documentation of the catalog or finding out from its developers the consensus meaning of the attribute.

Another problem arises when the interdependencies among attributes are not represented. For example, the duration of an event is related to the start time and the end time of that event, and in this example `execution-time`, `begin-execution-time` and `end-execution-time` are related. If some of the data is missing the `execution-time` it could be derived from its `start-execution-time` and the `end-execution-time`. Otherwise, queries would need to be formulated to include both cases explicitly. The advantage of having these kinds of relationships expressed declaratively as part of the metadata definitions is that queries only have to mention what is needed and disregard what particular metadata attributes are present in each specific case.

All these problems are exacerbated when there is a need to query several metadata catalogs, where the attribute names will be likely to be independent, and any correlation that might exist among attributes in different catalogs is not declared explicitly.

3 Standards for Semantic Representation and Reasoning

In this work we draw upon three semantic web technologies: semantic data representations defined using relevant domain terms, ontologies that attribute meaning to the terms and reasoners that can answer queries about the terms and their relationships.

RDF [6] is a web standard for representing resources on the web. It restricts the description of resources to statements composed of subject, predicate and object triples. It uses XML [7] as the interchange syntax. An RDF Schema (RDFS) [8] defines the terms that will be used in the RDF statements and gives specific meanings to them. The Web Ontology Language (OWL) may also be used to define those terms using more expressive representations. OWL builds on top of RDF. Because these languages are built on web standards, they take advantage of namespaces and URIs to define the scope of the definitions and to import definitions from distributed locations respectively.

We use OWL-DL representations in this work because of its expressive power and because there are efficient reasoners already available for it. We will briefly illustrate here the expressivity of OWL with examples from temporal reasoning, using definitions from the a variant of the OWL-Time ontology [9]. An interval can be defined as:

```

<owl:Class rdf:ID="IntervalThing">
<rdfs:subClassOf rdf:resource= "#TemporalThing"/>
<rdfs:subClassOf><owl:Restriction>
  <owl:onProperty rdf:resource="#from" />
  <owl:maxCardinality
rdf:datatype="&xsd;nonNegativeInteger">1</owl:maxCardinality>
  </owl:Restriction></rdfs:subClassOf>
  <rdfs:subClassOf><owl:Restriction>
  <owl:onProperty rdf:resource="#to" />
  <owl:maxCardinality
rdf:datatype="&xsd;nonNegativeInteger">1</owl:maxCardinality>
  </owl:Restriction></rdfs:subClassOf>
</owl:Class>

```

Here we define an `IntervalThing` as a subClass of a generic `TemporalThing` that has a beginning (“from”) and an end (“to”). It also defines restrictions on its properties “from” and “to” (declared below) in that they can only have one value (indicated by a cardinality of 1). Note that the prefixes owl, rdfs, and rdf refer to terms from their respective namespaces.

```

<owl:ObjectProperty rdf:ID="from">
  <rdfs:domain rdf:resource= "#TemporalThing"/>
  <rdfs:range rdf:resource= "#InstantThing"/>
  <rdf:type rdf:resource= "&owl;FunctionalProperty"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="to">
  <rdfs:domain rdf:resource= "#TemporalThing" />
  <rdfs:range rdf:resource= "#InstantThing"/>
  <rdf:type rdf:resource= "&owl;FunctionalProperty"/>
</owl:ObjectProperty>
<owl:DatatypeProperty rdf:ID="duration">
  <rdfs:domain rdf:resource= "#TemporalThing"/>
  <rdfs:range rdf:resource= "&xsd;duration"/>
</owl:DatatypeProperty>

```

The above definitions declare the properties “from”, “to”, and “duration” of any `TemporalThing`. Of note is the range of these properties. The range of a property implies the types of values that the property might have. The range of duration is a native XML Schema datatype called duration. The “&xsd” marker indicates the namespace of the XML Schema. XML Schema provides several built-in datatypes such as Integer, String, date`Time`, etc. More details and a thorough introduction of OWL, are available from. Many tools are available for OWL including ontology editors, parsers, and reasoners, many surveyed in [4].

It is important to note that OWL does not have the full expressive power of first-order logic, which is the representation of choice in many knowledge representation and reasoning systems. For example, the time ontology OWL-Time was originally specified in first order logic, and the OWL versions of it lack many of the axioms and constraints of the original. To address this issue, rule languages are being developed to complement OWL and to support the representation of more expressive relations and constraints. The following rule expresses how to derive the “to” property from the “from” property and “duration” property:

```

[r1: (?x rdf:type tme:IntervalThing), (?x tme:from ?a), (?x tme:duration ?t2),
(?a tme:at ?t1), sum(?t1, ?t2, ?t3)
  makeTemp(?v) -> (?v rdf:type tme:InstantThing) (?v tme:at ?t3) (?x tme:to ?v)]

```

There are no current semantic web standards for rule languages or query languages, although some have been proposed (RuleML[10] and OWL-QL[11]). The

rule format shown in the example above is used by Jena [12], the reasoner used in our system. Standard rule and query languages will inevitably emerge soon.

4 Approach

Figure 1 illustrates our approach. We propose to augment metadata catalogs with a semantic layer that supports queries in terms of virtual metadata attributes, resulting in virtual metadata catalog services. These attributes are virtual in that they are not really used in the implementation of the catalog. However, virtual metadata attributes can be used to query the catalog transparently as if they actually were associated with the data. To support this functionality, the virtual metadata attributes need to be mapped to the metadata attributes that are actually contained in the catalog (*actual attributes*).

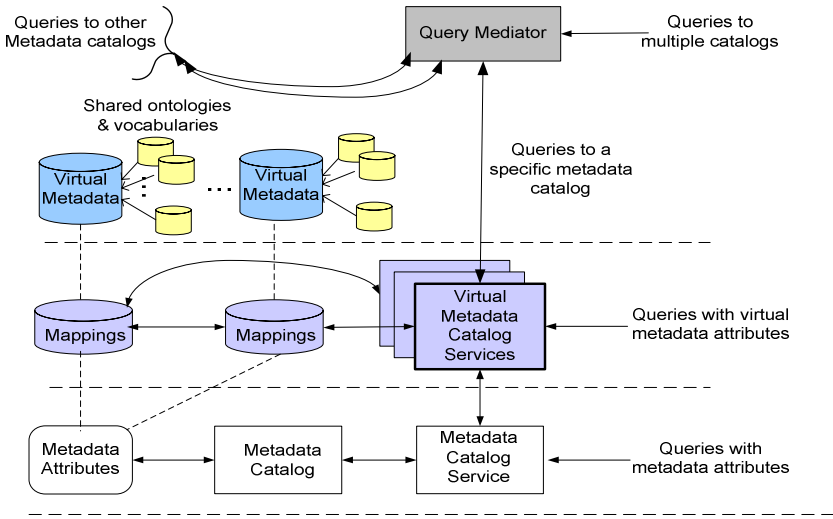


Fig. 1. Approach towards a distributed metadata catalog query

Mapping queries given in terms of virtual metadata attributes into queries in terms of the actual metadata attributes can be very complex. As we motivated in the examples of the previous section, expressive semantic representations and reasoners are needed to do these mappings automatically. We use standards for semantic representation and reasoning when they exist. Additional standards and tools are under development that will support increasingly more expressive query languages and mappings.

The figure also illustrates that in different contexts users may have different preferences or standards for querying the data in a catalog, resulting in alternative virtual metadata catalog services that can be built on top of the same underlying metadata catalog. For example, a catalog of historical weather data could be used by a climatologist to test a weather prediction model, or by an oceanographer to correlate underwater vegetation with weather conditions. Some of these virtual metadata

attributes could be drawn from shared ontologies or standard vocabularies, which are becoming commonplace in many scientific communities [13-17]. Many scientific ontologies are already undergoing conversion to semantic web standards [4, 6-8] and others will soon follow as the benefits of these expressive languages are shown. Users may also define their own virtual metadata attributes by creating customized ontologies, effectively creating personalized metadata catalogs.

The approach is modular and decentralized in that each virtual metadata catalog service reasons about its own virtual metadata within its own reasoners. This is in contrast with our work on Artemis, where a centralized reasoner resolved all the mappings to all the metadata catalogs. The advantages of this decentralized approach is that it will be more robust to failures. In addition, the reasoning tasks will be more manageable and will scale better as more metadata catalogs are added.

5 Virtual Metadata Catalogs

Figure 2 illustrates the architecture of our implementation of a virtual metadata catalog developed using MCS. We use OWL in combination with rules to express the query, the shared domain ontologies, and the virtual metadata attributes and mappings. The original query is provided as an OWL document that includes references to the domain ontologies from where the virtual metadata attributes in the query are drawn. The query may also reference terms from a generic catalog ontology that we have created. The purpose of this ontology is to define terms such as “files”, “views”, “collections”, that are used in typical queries to MCS.

The central component of the architecture is the Query Mapping module. It takes the OWL query and turns it into a MCS query that uses the metadata attributes that actually appear in the catalog. The MCS query is then submitted to the MCS, which returns all the references to data stored in it that satisfy the query. We will use an example to explain in detail how the Query Mapping module works.

Consider a query for data within a temporal interval starting on 10th October 2004 at 10am and a duration of 30 seconds. Suppose the user wishes to query using the virtual metadata attributes “from” and “duration”, both taken from the OWL Time ontology. Assume that the metadata attributes present in the MCS are “startDate” and “endDate”. The core of the original OWL query is:

```
<tme:IntervalThing rdf:ID="Interval1">
  <tme:from rdf:resource="#T1"/>
  <tme:duration rdf:datatype = "&xsd;duration"> PT30S </tme:duration>
</tme:IntervalThing>
<tme:InstantThing rdf:ID="T1">
  <tme:at rdf:datatype="&xsd;dateTime">2004-01-01T10:00:00</tme:at>
</tme:InstantThing>
```

The Virtual Metadata Attributes and Mappings express that the MCS “startDate” attribute is equivalent to the “from” virtual metadata attribute, and the MCS attribute “endDate” is equivalent to the “to” virtual metadata attribute. Here is how these mappings are specified:

```
<owl:ObjectProperty rdf:ID="startDate">
  <owl:equivalentProperty rdf:resource = "&tme;from"/>
  <terms:hasMCSAttribute> startDate </terms:hasMCSAttribute>
  <terms:pathToData->at</terms:pathToData>
```

```

</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="endDate">
  <owl:equivalentProperty rdf:resource = "&tme;to"/>
  <terms:hasMCSAttribute> endDate </terms:hasMCSAttribute>
<terms:pathToData>->at</terms:pathToData>
</owl:ObjectProperty>

```

The Query Mapping component accepts the OWL Query document and configures the semantic reasoner by loading the OWL ontologies and rules referenced in it. The query mapping process is depicted in Figure 3, and is composed of three major steps. First, the basic query constituents are created by running the reasoner and generating attribute/value pairs based on the rules defined. In our case, the rule that defines “to” in terms of interval duration would be used to generate the following attribute/value pairs:

```

{from=[2004-01-01T10:00:00 ^http://www.w3.org/2001/XMLSchema#dateTime],
 to=[2004-01-01T10:00:30Z ^http://www.w3.org/2001/XMLSchema#dateTime]}

```

The suffix starting with ^^ in RDF signifies the datatype of the value that it is appended to.

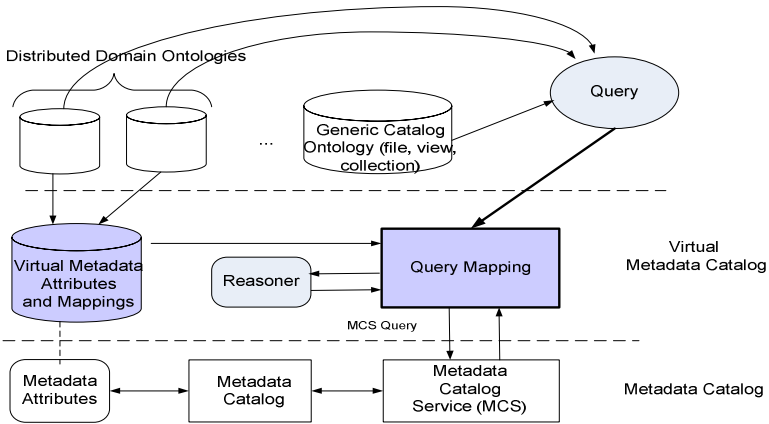


Fig. 2. Architecture of a Virtual Metadata

These virtual metadata attribute value pairs are then converted into MCS attribute value pairs by selecting the relevant subset of the triples and applying the relevant mappings. Therefore, “from” is replaced by “startDate”, and “to” is replaced by “endDate”. Another mapping performed in this step is the conversion of the values from the XML Schema Datatypes to the ones that are expected by the database. In our case, the dateTime formats of the OWL Query need to be converted to the Date types that are expected for the startDate and endDate. Finally, the MCS query is constructed by adding the operators to construct the appropriate query formula.

In our implementation we used data from three different domains: climate modeling, earthquake science, and workflow execution tracking. The climate modeling catalog contains such information as longitude, latitude, temperature and date and time. The earthquake science catalog collects data about simulation results that show seismic wave propagation over time. This catalog has more than one

hundred metadata attributes. Finally, the workflow tracking catalog includes data about the names of the workflow tasks, their execution duration, the execution location (resource used), success or failure and others. Although these various domains deal with different types of data, they all make use of temporal concepts.

To add new virtual metadata attributes, a user would only have to define their mappings into MCS attributes using similar definitions to the ones shown above to map "from" and "to" to "startTime" and "endTime" respectively.

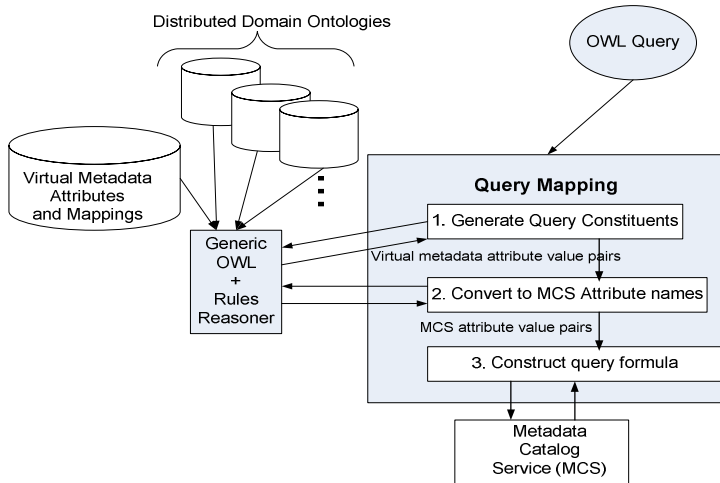


Fig. 3. A more detailed overview of the query mapping process

6 Discussion

Our work illustrates how semantic representations can be used to support virtual metadata attributes and how reasoners can be used to resolve queries that use them, opening the way for virtual metadata catalog services. As proof of concept, the system implemented so far can answer queries but that is only part of the functionality of a metadata catalog service. We plan to extend its functionality to the fullest in the future and include useful functions such as returning lists of available metadata attributes, publish data, group data into hierarchical collections, and set authorization information on objects.

An important issue that needs to be addressed more fully in the future is handling alternative data formats. In our examples, time points and durations were represented using the XML schema data types, but other metadata attributes may be defined using non-standard formats. A point in time can be expressed in different syntactic formats such as 2004/01/30-23:34:48 or as 30/1/04.11:34pm, and a query on time points for November 30 of 2004 should retrieve both. Addressing this issue requires representing mappings between alternative formats, which could be done using the same approach we have used to define mappings among different terms in our system.

The same issue applies to transport formats, for example the year 2004 could be rendered as a string or a number. At this point it is not clear how far reaching the transformations need to be. One can imagine a whole spectrum from simple date and time transformations to more complicate coordinate transformations as is sometimes necessary in astronomy. Astronomers for example use a variety of different coordinate formats to point to specific locations in the sky. Sometimes they also use a stellar object name to denote a location. We also plan to investigate how to support the integration of multiple catalog services using query mediators as was done in our previous work on Artemis. When catalogs are mapped to identical shared ontologies, their integration to a query mediator should be simplified.

Another important issue to investigate is how this architecture scales up to large amounts of attributes and multiple metadata catalog services. Semantic web technologies are being developed very quickly to reason efficiently as the amount of data and definitions grow. Existing systems handle millions of RDF triples, a basic unit to render RDF and OWL descriptions, however evaluating systems that rely on these technologies in real deployments.

7 Related Work

This work bridges several research areas: metadata management, query mediation and semantic web technologies. In this section we mention the most relevant work in these areas. In terms of metadata management, besides the Metadata Catalog Service mentioned in Section 1, the Storage Resource Broker (SRB) from the San Diego Supercomputing Center [18] and its associated MCAT Metadata Catalog [19] provide metadata and data management services. SRB supports a logical name space that is independent of physical name space. The logical objects, logical files in the case of SRB, can also be aggregated into collections. SRB provides various authentication mechanisms to access metadata and data within SRB. Unlike the work presented here, SRB is based on a centralized metadata catalog and does not provide semantic information about the catalog content.

In [20], the authors describe a mediator-based system that utilizes the semantics of the data exported by the data sources to integrate the data. A key assumption in the [20] paper is that the data sources export the semantics of the data. This work is complimentary as it provides a means for the semantics to be added to the data sources and thus available to existing mediator systems. The myGrid project [21] is developing and exploiting semantic web technology to describe and integrate a wide range of services in a grid environment. Data sources are modeled as semantic web services, and are integrated through web service composition languages. The result is a workflow that may include not only steps to access to data sources, but also as simulation or other data processing steps. A key difference between myGrid and the work presented here is that myGrid relies on the use of standard ontologies from the bioinformatics domain and thus the problem of semantic information representation is greatly simplified. Unlike other domains, scientists in bioinformatics have made great strides in design common semantic representations.

8 Conclusions

We have designed and implemented a virtual metadata catalog that provides rich semantic information about the catalog content in a variety of semantics views. The views are customized to a particular global ontology. This system provides an easy way for users to publish and discover data using metadata attributes that are appropriate for them. In this work we drew upon data from three different disciplines: atmospheric sciences, performance databases and earthquake science. We also put the Virtual Metadata Catalog in a broader context of a distributed system, where multiple such catalogs would exist and query mediation technologies such as those based on our previous system (Artemis) would be used to query across the multiple catalogs.

In future work, we would like to extend the query mapping process to make it more robust and better integrated with OWL reasoners as well as the MCS back end. We would like to formalize the mappings of different query expressions in a comprehensive framework. This will be facilitated as standard OWL query languages emerge. We also plan to use the virtual metadata catalog in some of our ongoing projects.

Acknowledgements

We gratefully acknowledge the support from the NSF's Shared Cyberinfrastructure program under grant SCI-0455361 and the NSF SCEC-CME project with grant number EAR-0122464.

References

- [1] E. Deelman, et al. Representing Virtual Data: A Catalog Architecture for Location and Materialization Transparency," TR GriPhyN-2001-14, 2001.
- [2] P. Messina and A. Szalay, "Building the Framework for the National Virtual Observatory," <http://www.us-vo.org/docs/nvo-proj.doc>, 2002.
- [3] R. Tuchinda, S. Thakkar, Y. Gil, and E. Deelman., "Artemis: Integrating Scientific Data on the Grid," Proceedings of IAAI, San Jose, California, 2004 .
- [4] "Web Ontology Language (OWL)," <http://www.w3.org/2001/sw/WebOnt/>
- [5] E. Deelman, et al. "Grid-Based Metadata Services," Proceedings of Statistical and Scientific Database Management (SSDBM), Santorini, Greece, 2004.
- [6] "RDF," <http://www.w3.org/TR/REC-rdf-syntax/>,
- [7] "XML Schema," <http://www.w3.org/XML/Schema>
- [8] "RDF Schema," <http://www.w3.org/TR/rdf-schema/>
- [9] J. R. Hobbs and F. Pan, "An Ontology of Time for the Semantic Web," ACM TALIP: Special issue on Temporal Information Processing, (3,) pp. 66-85, 2004.
- [10] "RuleML," <http://www.ruleml.org/>
- [11] R. Fikes, P. Hayes, and I. Horrocks, "OWL-QL: A Language for Deductive Query Answering on the Semantic Web," KSL Technical Report 03-14 2003.
- [12] "Jena," <http://jena.sourceforge.net>
- [13] C. J. Wroe et al., "A Methodology to Migrate the Gene Ontology to a Description Logic Environment Using DAML+OIL," Proc. of Pacific Symposium on Biocomputing 2003.

- [14] U. Hahn and S. Schulz, "Building a Very Large Ontology from Medical Thesauri," in Handbook on Ontologies, R. S. S. Staab, Ed.: Springer, 2004.
- [15] "GO," <http://www.geneontology.org/>
- [16] "UMLS," <http://www.nlm.nih.gov/research/umls/>
- [17] J. Golbeck, et al. "The National Cancer Institute's Thesaurus and Ontology," Journal of Web Semantics, vol. 1, 2003.
- [18] C. Baru et al, "The SDSC Storage Resource Broker," Proceedings of CASCON'98, 1998.
- [19] "MCAT (Version 1.1)," <http://www.npaci.edu/DICE/SRB/mcat.html>
- [20] B. Ludäscher, et al, "A Model-Based Mediator System for Scientific Data Management," in Bioinformatics:Managing Scientific Data, Critchlow, Lacroix, Eds.: MK, 2003.
- [21] C. Wroe, et al, "A Suite of DAML+OIL Ontologies to Describe Bioinformatics Web Services and Data," Int. J. of Cooperative Information Systems, 2003.

Combining Provenance with Trust in Social Networks for Semantic Web Content Filtering

Jennifer Golbeck

University of Maryland, College Park, College Park MD 20742, USA

golbeck@cs.umd.edu

<http://mindswap.org>

Abstract. Social networks are a popular movement on the web. On the Semantic Web, it is simple to make trust annotations to social relationships. In this paper, we present a two level approach to integrating trust, provenance, and annotations in Semantic Web systems. We describe an algorithm for inferring trust relationships using provenance information and trust annotations in Semantic Web-based social networks. Then, we present an application, FilmTrust, that combines the computed trust values with the provenance of other annotations to personalize the website. The FilmTrust system uses trust to compute personalized recommended movie ratings and to order reviews. We believe that the results obtained with FilmTrust illustrate the success that can be achieved using this method of combining trust and provenance on the Semantic Web.

1 Introduction

Social Networks have become a popular movement on the web as a whole, and the Semantic Web is rich with social network information. Friend of a Friend (FOAF) is an OWL-based vocabulary for representing personal and social network information; data using FOAF makes up a significant percentage of all data on the Semantic Web. Within these social networks, users can take advantage of other ontologies for annotating additional information about their social connections. This may include the type of relationship (e.g. "sibling", "significant other", or "long lost friend"), or how much they trust the person that they know. Annotations about trust are particularly useful, as they can be applied in two ways. First, using the annotations about trust and the provenance of those statements, we can compute personalized recommendations for how much one user (the *source*) should trust another unknown user (the *sink*) based on the paths that connect them in the social network and the trust values along those paths. Once those values can be computed, there can be a second application of the trust values. In a system where users have made statements and we have the provenance information, we can filter the statements based on how much the individual user trusts the person who made the annotation. This allows for a common knowledge base that is personalized for each user according to who they trust.

In this paper, we will present a description of social networks and an algorithm for inferring trust relationships within them. Then, we will describe FilmTrust, a movie recommender system, where trust is used to filter, aggregate, and sort information.

2 Social Networks and Trust on the Semantic Web

Social networks on the Semantic Web are usually created using the FOAF vocabulary [2]. There are over 10,000,000 people with FOAF files on the web, describing their personal information and their social connections [4]. There are several ontologies that extend FOAF, including the FOAF Relationship Module [3] and the FOAF Trust Module [4]. These ontologies provide a vocabulary for users to annotate their social relationships in the network. In this research, we are particularly interested in trust annotations.

Using the FOAF Trust Module, users can assign trust ratings on a scale from 1 (low trust) to 10 (high trust). There are currently around 3,000 known users with trust relationships included in their FOAF profiles. Once that information is aggregated, we can make computations with trust values. We choose a specific user, and look at all of the trust ratings assigned to that person. With that information, we can get an idea of the average opinion about the person's trustworthiness. Trust, however, is a subjective concept. Consider the simple example of asking whether the President is trustworthy. Some people believe very strongly that he is, and others believe very strongly that he is not. In this case, the average trust rating is not helpful to either group.

In this work, we use the term "provenance" to refer to *who* made a particular statement. Since we have provenance information about the trust annotations in FOAF networks, we can significantly improve on the average case. If someone (the *source*) wants to know how much to trust another person (the *sink*), we can look at the provenance information for the trust assertions, and combine that with the source's directly assigned trust ratings, producing a result that weights ratings from trusted people more highly than those from untrusted people.

In this section, we present an algorithm for inferring trust relationships that combines provenance information with the user's direct trust ratings.

2.1 Background and Related Work

When two individuals are directly connected in the network, they can have trust ratings for one another. Two people who are not directly connected do not have that trust information available by default. However, the paths connecting them in the network contain information that can be used to infer how much they may trust one another.

For example, consider that Alice trusts Bob, and Bob trusts Charlie. Although Alice does not know Charlie, she knows and trusts Bob who, in turn, has information about how trustworthy he believes Charlie is. Alice can use information from Bob and her own knowledge about Bob's trustworthiness to infer how much she may trust Charlie. This is illustrated in Figure 1.

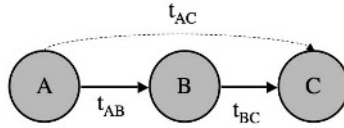


Fig. 1. An illustration of direct trust values between nodes A and B (t_{AB}), and between nodes B and C (t_{BC}). Using a trust inference algorithm, it is possible to compute a value to recommend how much A may trust C (t_{AC}).

To accurately infer trust relationships within a social network, it is important to understand the properties of trust networks. Certainly, trust inferences will not be as accurate as a direct rating. There are two questions that arise which will help refine the algorithm for inferring trust: how will the trust values for intermediate people affect the accuracy of the inferred value, and how will the length of the path affect it.

We present an algorithm for inferring trust relationships in social networks, but this problem has been approached in several ways before. Here, we highlight some of the major contributions from the literature and compare and contrast them with our approach.

The EigenTrust algorithm [7] is used in peer-to-peer systems and calculates trust with a variation on the PageRank algorithm[9], used by Google for rating the relevance of web pages to a search. EigenTrust is designed for a peer-to-peer system while ours is designed for use in humans' social networks, and thus there are differences in the approaches to analyzing trust. In the EigenTrust formulation, trust is a measure of performance, and one would not expect a single peer's performance to differ much from one peer to another. Socially, though, two individuals can have dramatically different opinions about the trustworthiness of the same person. Our algorithms intentionally avoid using a global trust value for each individual to preserve the personal aspects that are foundations of social trust.

Raph Levin's Advogato project [8] also calculates a global reputation for individuals in the network, but from the perspective of designated seeds (authoritative nodes). His metric composes certifications between members to determine the trust level of a person, and thus their membership within a group. While the perspective used for making trust calculations is still global in the Advogato algorithm, it is much closer to the methods used in this research. Instead of using a set of global seeds, we let any individual be the starting point for calculations, so each calculated trust rating is given with respect to that person's view of the network.

Richardson et. al.[10] use social networks with trust to calculate the belief a user may have in a statement. This is done by finding paths (either through enumeration or probabilistic methods) from the source to any node which represents an opinion of the statement in question, concatenating trust values along the paths to come up with the recommended belief in the statement for that

path, and aggregating those values to come up with a final trust value for the statement. Current social network systems on the Web, however, primarily focus on trust values between one user to another, and thus their aggregation function is not applicable in these systems.

2.2 Issues for Inferring Trust

We expect that people who the user trusts highly will tend to agree with the user more about the trustworthiness of others than people who are less trusted. To make this comparison, we can select triangles in the network. Given nodes n_i , n_j , and n_k , where there is a triangle such that we have trust values t_{ij} , t_{ik} , and t_{kj} , we can get a measure of how trust of an intermediate person can affect accuracy. Call Δ the difference between the known trust value from n_i to n_k (t_{ik}) and the value from n_j to n_k (t_{jk}). Grouping the Δ values by the trust value for the intermediate node (t_{ij}) indicates on average how trust for the intermediate node affects the accuracy of the recommended value. Several studies [11],[4] have shown a strong correlation between trust and user similarity in several real-world networks.

It is also necessary to understand how the paths that connect the two individuals in the network affect the potential for accurately inferring trust relationships. The length of a path is determined by the number of edges the source must traverse before reaching the sink. For example, source-sink has length two. Does the length of a path affect the agreement between individuals? Specifically, should the source expect that neighbors who are connected more closely will give more accurate information than people who are further away in the network? Previous work[4],[6] has also addressed this issue and shown that, as expected, shorter paths lead to more accurate information. As with trust values, it will be important to consider the length of connecting paths when developing an algorithm for inferring trust.

2.3 TidalTrust: An Algorithm for Inferring Trust

The effects of trust ratings and path length described in the previous section guided the development of TidalTrust, an algorithm for inferring trust in networks with continuous rating systems. The following guidelines can be extracted from the analysis of the previous sections:

1. For a fixed trust rating, shorter paths have a lower error ($\overline{\Delta}$).
2. For a fixed path length, higher trust ratings have a lower $\overline{\Delta}$.

This section describes how these features are used in the TidalTrust algorithm.

Incorporating Path Length. The analysis in the previous section indicates that a limit on the depth of the search should lead to more accurate results, since the $\overline{\Delta}$ increases as depth increases. If accuracy decreases as path length increases, as the earlier analysis suggests, then shorter paths are more desirable. However, the tradeoff is that fewer nodes will be reachable if a limit is imposed on the path depth. To balance these factors, the path length can vary from

one computation to another. Instead of a fixed depth, the shortest path length required to connect the source to the sink becomes the depth. This preserves the benefits of a shorter path length without limiting the number of inferences that can be made.

Incorporating Trust Values. The previous results also indicate that the most accurate information will come from the highest trusted neighbors. To incorporate this into the algorithm, we establish a minimum trust threshold, and only consider connections in the network with trust ratings at or above the threshold. This value cannot be fixed before the search because we cannot predict what the highest trust value will be along the possible paths. If the value is set too high, some nodes may not have assigned values and no path will be found. If the threshold is too low, then paths with lower trust may be considered when it is not necessary. We define a variable *max* that represents the largest trust value that can be used as a minimum threshold such that a path can be found from source to sink. *max* is computed while searching for paths to the sink by tracking trust values that have been seen.

Full Algorithm for Inferring Trust. Incorporating the elements presented in the previous sections, the final TidalTrust algorithm can be assembled. The name was chosen because calculations sweep forward from source to sink in the network, and then pull back from the sink to return the final value to the source.

$$t_{is} = \frac{\sum_{j \in \text{adj}(j) \mid t_{ij} \geq \text{max}} t_{ij}t_{js}}{\sum_{j \in \text{adj}(j) \mid t_{ij} \geq \text{max}} t_{ij}} \quad (1)$$

TidalTrust is a modified breadth-first search. The source's inferred trust rating for the sink ($t_{\text{source},\text{sink}}$) is a weighted average if the source's neighbors' ratings of the sink (see Formula 1).

The source node begins a search for the sink. It will poll each of its neighbors to obtain their rating of the sink. If the neighbor has a direct rating of the sink, that value is returned. If the neighbor does not have a direct rating for the sink, it queries all of its neighbors for their ratings, computes the weighted average as shown in Formula 1, and returns the result .

To improve the accuracy of the algorithm, path length and path strength considerations are included. At each node that is reached in the search, Each node that is reached performs this process, keeping track of the current depth from the source. Each node will also keep track of the strength of the path to it. Nodes adjacent to the source will record the source's rating assigned to them. Each of those nodes will poll their neighbors. The strength of the path to each neighbor is the minimum of the source's rating of the node and the node's rating of its neighbor. The neighbor records the maximum strength path leading to it. Once a path is found from the source to the sink, the depth is set at the maximum depth allowable. Since the search is proceeding in a Breadth First

Search fashion, the first path found will be at the minimum depth. The search will continue to find any other paths at the minimum depth. Once this search is complete, the trust threshold (*max*) is established by taking the maximum of the trust paths leading to the sink. With the *max* value established, each node can complete the calculations of a weighted average by taking information from nodes that they have rated at or above the *max* threshold.

The accuracy of this algorithm is addressed in depth in [4] and [6]. While the error will vary from network to network, our experiments in two real world social networks show the results to be accurate to within about 10%.

2.4 Accuracy of TidalTrust

As presented above, TidalTrust strictly adheres to the observed characteristics of trust: shorter paths and higher trust values lead to better accuracy. However, there are some things that should be kept in mind. The most important is that networks are different. Depending on the subject (or lack thereof) about which trust is being expressed, the user community, and the design of the network, the effect of these properties of trust can vary. While we should still expect the general principles to be the same—shorter paths will be better than longer ones, and higher trusted people will agree with us more than less trusted people—the proportions of those relationships may differ from what was observed in the sample networks used in this research. A more extensive comparison and analysis of accuracy, including a comparison to a PKI algorithm[1], is available in [6] and [4].

Table 1. $\bar{\Delta}$ for TidalTrust and Simple Average recommendations in both the Trust Project and FilmTrust networks. Numbers are absolute error on a 1-10 scale.

Algorithm		
Network	TidalTrust	Simple Average
Trust Project	1.09	1.43
FilmTrust	1.35	1.93

3 Using Trust to Personalize Content

While the computation of trust values is in and of itself a user of provenance and annotations together, the resulting trust values are widely applicable for personalizing content. If we have provenance information for annotations found on the semantic web, and a social network with trust values such that a user can compute the trustworthiness of the person who asserted statement, then the information presented to the user can be sorted, ranked, aggregated, and filtered according to trust.

FilmTrust, at <http://trust.mindswap.org>, is a website with a social network. Users can rate movies on a scale of 0.5 to 4 stars, and write reviews of films. While the users interact with a simple web interface, the data is all stored as Semantic

Web annotations. In the social network, users also rate the trustworthiness of their friends on a scale of 1-10 using the FOAF Trust Module.

The trust values are used in conjunction with the TidalTrust algorithm to present personalized views of movie pages. When the user chooses a film, they are presented with basic film data, the average rating of the movie, a personalized recommended rating, and the reviews written by users. The personalized recommended rating is computed by first selecting a set of people who rated the movie. The selection process considers trust and path length; details on how this set of people are chosen are provided in [5]. Using the trust values (direct or inferred) for each person in the set who rated the movie as a weight, and computing the weighted average rating. For the set of selected nodes S , the recommended rating r from node s to movie m is the average of the movie ratings from nodes in S weighted by the trust value t from s to each node:

$$r_{sm} = \frac{\sum_{i \in S} t_{si} r_{im}}{\sum_{i \in S} t_{si}} \quad (2)$$

We tested the quality of these results in FilmTrust by comparing the trust-based rating with the known rating that a user gave to a movie. While the experimental details are beyond the scope of this paper, the results were encouraging. We have shown in [4] and [6] that in the FilmTrust system, recommended ratings produced with trust are significantly more accurate than the simple average ratings as well as recommended ratings generated using a Pearson correlation-based automated collaborative filtering algorithm when the user's opinion of a movie is at least 1 star different from the average.

Trust values for users in the system are also used to order movie reviews. When there are multiple reviews for a movie, the reviews from the most trusted users are displayed first. Thus, information from people the users trust is displayed more prominently. A small user study[4] showed a strong user preference for this ordering, because the most relevant information for the user was easiest to see.

4 Conclusions and Future Work

In this paper, we have presented a two level approach to integrating trust, provenance, and annotations in Semantic Web systems. First, we presented an algorithm for computing personalized trust recommendations using the provenance of existing trust annotations in social networks. Then, we introduced two applications that combine the computed trust values with the provenance of other annotations to personalize websites. In FilmTrust, the trust values were used to compute personalized recommended movie ratings and to order reviews. We believe the FilmTrust system offers promise for using trust systems for additional content filtering. We envision social networks with trust values being incorporated to more critical systems to judge statements. We are currently working to combine a social network with Profiles In Terror¹, an open source intelligence

¹ <http://profilesinterror.mindswap.org>

project. Intelligence professionals can assign trust based on how much they trust the information and analyses provided by other users. That, in turn, can be used with provenance about the statements to rate the quality of information in the system.

Acknowledgments

This work, conducted at the Maryland Information and Network Dynamics Laboratory Semantic Web Agents Project, was funded by Fujitsu Laboratories of America – College Park, Lockheed Martin Advanced Technology Laboratory, NTT Corp., Kevric Corp., SAIC, the National Science Foundation, the National Geospatial-Intelligence Agency, DARPA, US Army Research Laboratory, NIST, and other DoD sources.

References

1. T. Beth, M. Borchering, and B. Klein. Valuation of trust in open networks. *Proceedings of ESORICS 94.*, 1994.
2. D. Brickley and L. Miller. Foaf vocabulary specification. <http://xmlns.com/foaf/0.1/>, 2005.
3. I. Davis and E. V. Jr. Relationship: A vocabulary for describing relationships between people. 2004.
4. J. Golbeck. *Computing and Applying Trust in Web-based Social Networks*. Ph.D. Dissertation, University of Maryland, College Park, 2005.
5. J. Golbeck. Filmtrust: Movie recommendations using trust in web-based social networks. *Proceedings of the Consumer Communication and Networking Conference*, 2006.
6. J. Golbeck. Generating Predictive Movie Recommendations from Trust in Social Networks. *Proceedings of The Fourth International Conference on Trust Management*, 2006.
7. S. D. Kamvar, M. T. Schlosser, and H. Garcia-Molina. The eigentrust algorithm for reputation management in p2p networks. *Proceedings of the 12th International World Wide Web Conference*, May 20-24, 2004.
8. R. Levin and A. Aiken. Attack resistant trust metrics for public key certification. *7th USENIX Security Symposium*, 1998.
9. L. Page, S. Brin, R. Motwani, and T. Winograd. The pagerank citation ranking: Bringing order to the web. *Technical Report 1998, Stanford University*, 1998.
10. M. Richardson, R. Agrawal, and P. Domingos. Trust management for the semantic web. *Proceedings of the Second International Semantic Web Conference*, 2003.
11. C.-N. Ziegler and J. Golbeck. Investigating Correlations of Trust and Interest Similarity. *Decision Support Services*, 2006.

Recording Actor State in Scientific Workflows*

Ian Wootten, Omer Rana, and Shrija Rajbhandari

School of Computer Science, Cardiff University, UK

Abstract. The process which leads to a particular data item, or its provenance, may be documented in a number of ways. The recording of actor state assertions – essentially data that a client or service actor may assert about itself regarding an interaction, is evaluated as a critical provenance component within a service-oriented architecture. Actor state data can be combined with assertions of interaction to enable better reasoning within a provenance system. The types of data that may be recorded as actor state are subjective, and dependent on the nature of the application and the eventual use that is likely to be made of this data. A registry system that allows monitoring tools to be related to user needs is described with reference to an application scenario.

1 Introduction

The provenance of a piece of data is the process that led to that piece of data [1,2]. Typically, with service based projects, concern is primarily held with documenting the interaction between clients and services (actors) which were involved in a particular process as a means of capturing data provenance. Groth *et al.* [1] outline that for some data, its provenance is represented by some suitable documentation of process which led to it. This documentation includes in part the internal states of actors within the context of a particular interaction. Here, we make clear our definition of actor state data:

Actor State Data: That information regarding the state of an actor in the context of a specific interaction. A single assertion of actor state may concern the internal flow of data involved in interaction results within an actor, or the hosting environment state at a particular point in time. Assertions of actor state can only be recorded by the actor whom the data is about and not by a workflow enactment engine. An actor must therefore explicitly decide to make available such information to third parties. Our focus within this paper is primarily on how such actor state information can enhance documentation of interaction.

To capture the documentation of interaction between actors involved in a particular process, *interaction assertions* may also be used. Such assertions specify which actors are involved and the messages exchanged between them. Interaction assertions may be recorded by a workflow enactment engine (by copying all messages exchanged between actors in a workflow), or it may be explicitly recorded by the actors themselves.

* This research is funded in part by EPSRC PASOA project GR/S67623/01.

The provenance of data is concerned with how we arrive at a particular data item, and assertions of actor state provide valuable information on how a particular actor state (for an involved actor) has been reached during the creation of that data item. This is achieved through the documentation of transformations made upon the input data within an actor, and the state the system (upon which an actor is hosted) was in when those transformations were made. Through capturing assertions regarding the transformations on input data we are able to determine what functionality an actor was invoking, i.e. what an actor was doing. Capturing the state of the system also records the context under which an actor was operating. Exposing this context allows insight into what conditions were set which could affect the overall data result.

Actor state assertion recording differs from interaction assertion recording – which may be recorded by a third party. All assertions concerning actor state become unverifiable if made by a third party, and as such the actor is the only party able to assert its own state.

Using actor state data it is possible to evaluate the behavior of an actor over the past and make predictions on its likely future behavior. Coupled with interaction assertions, such data may be used to evaluate whether a particular actor is the cause of an error or inaccurate result within a workflow instance. It also allows a better understanding of the performance patterns observed upon an actor, through using interaction assertions to determine what was being done when a particular behaviour pattern occurred. Within the context of a SOA, the description of internal flow of data within an actor would also constitute actor state data. An actor may make public the internal functions which were performed on the input data to obtain output data. From these functions, it is possible to construct a directed acyclic graph (DAG) detailing the process performed internally within an actor.

We attempt to develop a customisable architecture for the recording of actor state assertions. Familiarity with the concepts of Web Services and SOAs is assumed. The rest of this paper is organized as follows: section 2 provides requirements for an architecture which supports actor state assertion recording. Section 3 contains the types of data which may be recorded by an actor, and ways in which we may categorize such data. Section 4 presents an architecture to satisfy the requirements identified previously and an evaluation of a prototype system based on the architecture is provided. A summary of related work is given in section 5 and our concluding remarks are given in section 6.

2 Requirements

A provenance recording system must address a number of requirements, especially when used in the context of a SOA. The requirements for the storage of provenance information within a SOA have been highlighted in [1,2,3]. Here we identify requirements which influenced our prototype design:

An actor state assertion system should record the internal data flow within an actor in a verifiable, reproducible manner: It is possible to identify the activities undertaken by an actor as a DAG – which describes a series of transformations performed on some input data. Such graphs allow identification of how a particular data item was produced when followed in reverse. Hence, an actor that receives input X could transform it through a series of functions $f_i()$, eventually leading to an output Y . This can be achieved through instrumentation of that actor with relevant recording hooks describing such transformations and subsequent collection of this information by a provenance system.

An actor state assertion system should record in a non-repudiable manner any data generated by an actor: It is necessary to assume that information collected from any monitoring sources is accurate and provides a true reflection of the state of the actor. As sources are co-located with an actor this assumption should usually be satisfied (although the recording accuracy may differ between sources).

An actor state assertion system should be scalable, general and customisable: As previously stated in section 1, an actor needs to be viewed as a complete entity for actor state assertion, i.e. an actor asserts its own state. There exist a variety of mechanisms for capturing state information, and as such the available tools across potential applications differ significantly. It is therefore necessary for an actor state assertion system to be able to be customised to cope with the variety of information sources and the platforms upon which it will be hosted. A pertinent question here relates to what data needs to be recorded and at what frequency. Once again, both of questions can only be answered when we identify such a systems' application domain.

3 Actor State Assertion Categories

Actor state assertion categories describe the types of actor state data that may be recorded. For each description, we use the term *node* to describe the system on which an actor is hosted, and *lifetime* to refer to the length of time for which the actor (client or service) is available. Persistent actors may be classified as having an infinite lifetime.

Static: That data which does not change throughout the lifetime of an actor. As a result, static data need only be recorded once during process execution. Such data items have been previously investigated, and include: (i) Per-Node: node identity, operating system, etc.; (ii) Per-Actor: actor identity, name, owner, version, capability, etc. Such information is similar to that published by an actor in a registry service in a SOA.

Dynamic: That data which may change during the lifetime of an actor. It is therefore necessary to record this data at periodic intervals over the lifetime of an actor. We assume that actors within our architecture do not maintain a persistent state. Such data items may include: (i) Per-Node: memory usage, network traffic,

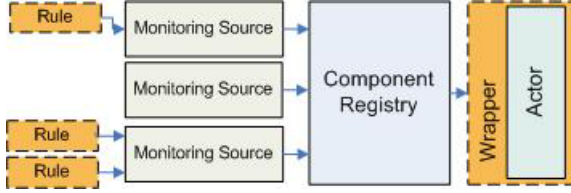


Fig. 1. Component Registry

etc. Such information needs to be recorded by the platform hosting the actor, and may be made available on demand. The accuracy of such dynamic data is dependent on the type of measurement tools being used; (ii) Per-Actor: service execution time, uptime, availability, etc. Such dynamic data is usually derived from other, less complex recorded metrics.

4 Conceptual Architecture

A registry based architecture for recording actor state is presented motivated by the requirements outlined in section 2. An end user may indicate which data is likely to be most significant to them – generally via the use of a configuration file. The basic architecture for a registry is given in figure 1. A component registry is co-located with an actor and holds details of the monitoring sources which are available on the platform hosting the actor. We consider the simplest case of one actor per platform in the first instance. The registry contains a description of interfaces through which monitoring sources may be contacted, and a mechanism to specify the time at which such requests may be scheduled.

A **monitoring source** may be a provider of a single piece of information or a number of metrics, and therefore a way of distinguishing the relevant useful information returned from the source becomes necessary. Within the registry, a number of **rules** may be associated with a single monitoring source, and specify the recorded actor state metrics desired during an actors’ lifetime. Two types of rules exist within our architecture: $R_A(e)$ are **runtime rules** – and may be triggered to be executed by an external or actor administrator generated event e , e.g. a service invocation request. Such rules are immediately scheduled to execute when an event of type e is detected by an actor. This provides functionality to an end user who may only want to record actor state whilst a service is being invoked. $R_B(t_1, t_2)$ are **scheduled rules** which execute between a time interval $(t_2 - t_1)$ – where t_i is based on the actor clock. Using R_B it is possible to record the state of an actor outside the context of any particular interaction. Rules of type R_B must be defined and managed by an actor administrator, and cannot be accessed via a third party. For an actor that is long running, t_1 may correspond to the time when the actor was started, and t_2 set to a large value. The result produced as a consequence of running rules of type R_A and R_B may be: (i) raw data – in this case monitored data over the particular

period in question is returned as an array of values. The data corresponds to the raw data from the monitoring tool being used; (ii) interval data – in this case the output from the monitoring tool is sampled at periodic intervals defined by an actor administrator. An array of values is returned, corresponding to a value at each sample point; (iii) aggregate data – in this case only a single value for a particular metric is returned. Such a value may correspond to either the min, max or average over the particular period. An end user must therefore register their rules with the actor administrator if they require particular actor state information to be monitored. The presence of a registry allows re-use of rules between end users. Due to the tree-like hierarchy of rule-monitor-registry associations, the configuration of a registry is achieved through an XML file.

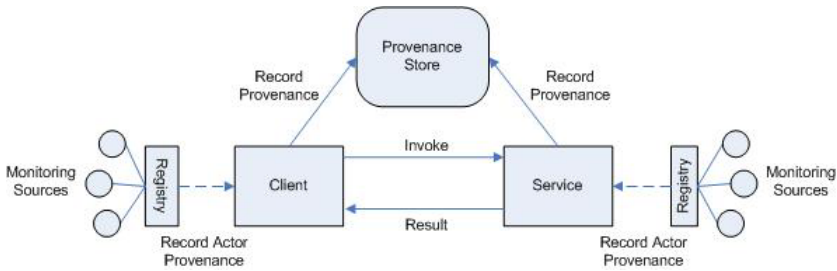


Fig. 2. Actor State Recording within a SOA using PReP [3]

Figure 2 shows how it is possible to capture both interaction and actor assertions within a SOA. The figure illustrates how actor assertions may be related to interaction assertions by extending the data that is submitted via the Provenance Recording Protocol (PReP) [3] to an external Provenance Store (a public and possibly remote repository). Both the client and service within such an architecture would have independent registries, containing references to locally available monitoring sources. Every time an interaction occurs between a client and service, each would submit their view of their interaction to a mutually agreed upon store.

Using Actor State Assertions: in a Web Service based workflow, a scientist does not have direct access to the system hosting the service actors. Using actor state information, such a user can make inferences about how or why a workflow has performed in a particular way. For example, consider a provenance-enabled workflow comprising of two services a and b , the workflow is executed twice to yield two sets of interaction assertions i_1 and i_2 in a Provenance Store. It also asserts actor state records a_1, a_2 and b_1, b_2 for each service actor. Both executions yield the same results, but the time of execution differ significantly. On inspection of i_1 and i_2 , the experimenter has no means of determining the source of such a problem, due to only the data which has passed between a and b being recorded – as both interact in the same manner. Inspection of the actor’s

state record a_2 reveals performance metrics indicating high usage of a 's resources whilst being invoked. While b 's performance shows no difference between b_1 and b_2 , it is possible for the scientist to conclude that a was the most likely source of such a discrepancy. If the DAG used within an actor is also known, this can be used in conjunction with the recorded data to locate the function within an actor which may have been the source of a particular problem.

4.1 Implementation

A prototype of the architecture described in section 4 has been realised, and actor state recorded using an SOA-based approach. The Ganglia system (<http://ganglia.sourceforge.net>) has been used as a monitoring tool for obtaining metrics relevant to node state and recorded during service execution. Rules are described within our registry, along with information about how often they are to be executed and which monitoring sources they are associated with through configuration files expressed in XML. An actor state is recorded using rules associated with the registered monitoring source to a local provenance store. A coordinator process is responsible for checking which rules are valid at any given time.

In figure 3 we describe a rule written in XQuery which returns results from a Ganglia XML document. Within the query, `$ganglia:doc` refers to the document we are querying, which is bound to the XQuery at execution time. Using this description, we return the integer value (indicated in figure 3 as *VAL*) of the number of bytes in per second (`bytes_in`) (KB/s), packets in per second (`pkts_in`) and maximum transmission rate of the network (`mtu`). These values are then operated upon as indicated in figure 3 to determine the current (network) throughput of this actor.

```
let $N := data($ganglia:doc//METRIC[NAME="bytes_in"]/VAL)
let $X := data($ganglia:doc//METRIC[NAME="pkts_in"]/VAL)
let $R := data($ganglia:doc//METRIC[NAME="mtu"]/VAL)
let $bpp := $N div $X          (: Calculate number of Bytes Per Packet :)
let $tp := ($bpp*$N) div $R

return <metrics><throughput>{$tp}</throughput></metrics>
      (: Return our result :)
```

Fig. 3. Example XQuery Rule Implementation

4.2 Evaluation

For evaluation, the registry prototype is used to record assertions regarding actor state during invocation of a data modeling service [4]. The modeller has a number of data processing techniques and neural network and statistical models which take incoming data sets from a client and generate models based upon them. There are a number of modeling algorithms exposed which vary the accuracy

of the model produced, possibly at the expense of incurring a greater computation time. Our experiments use Quantitative Structure-Activity Relationship (QSAR) to attempt to correlate biological activity to chemical compound structure described in the data set sent to it.

All experiments were conducted on a Ubuntu Linux System – AMD processor running at 1.83GHz with 512Mb of RAM. Both service and client actors are located on the same system, and for each experiment the service actor is invoked 100 times with input data sets of varying sizes, with the length of time of invocation recorded. Experiments are conducted for rules scheduled to be recorded simultaneously (at 1000ms intervals). A Ganglia monitoring daemon is installed making system metrics available as XML documents. This daemon is described as the monitoring source within the registry. The registry contains configuration files to enable scheduling of rules for runtime invocation, with each rule reflecting a single metric available through Ganglia. The results of rule execution are recorded to a local file system.

For our preliminary benchmark, we note that the time taken to invoke the service when no assertions are made, upon a 34KB data set is approximately 1404ms. On recording rules scheduled with intervals of 1000ms, it is noted that the trend for overall time for execution of the service increases on addition of each rule, reaching a maximum of 38062ms for 5 rules. We can see by these results, that while our prototype is able to record actor state assertions, it incurs a significant overhead against a non-asserting actor operating on the same data set. Making the comparison against our initial requirements, whilst the prototype has been produced in a general and customisable manner, actors where large amounts of state data may be produced will evidently suffer from a performance degradation, especially where large rule sets are constructed. Further work therefore is necessary to modify our system to enable it to be scalable to such situations, such as the use of a cache for monitoring source data and its asynchronous capture.

Table 1. Average Time to Complete Service Invocation

No Of Rules	Size of Data Set (KB)									
	34	68	102	136	170	204	238	272	306	340
0	1404	3338	5604	8795	10382	15223	14422	23309	18114	24613
1	1752	3863	6233	8574	11220	13383	17154	18142	21574	26046
5	2416	5680	7686	15134	15826	15979	23082	28531	29919	38062

5 Related Work

In SOAs there are a number of requirements necessary for the capture of the provenance of interactions [1,2,3]. The PASOA project (<http://www.pasoa.org/>) has highlighted a method of representing assertions made about processes by the actors involved through use of a p-assertion [1], suggesting 3 different p-assertion types (Interaction, Relationship and Actor State). The p-assertion presents a

possible manner in which to represent assertions in our local provenance store and is already being used within other projects. At the German Aerospace Center for instance, p-assertions are being used in order to capture actor state elements such as computation completion states (crashed, interrupted etc) and workflow configuration parameters in the simulation of complex flight manoeuvres [5]. Such capture achieves a confidence in simulation results which a non actor state recording mechanism may not.

A number of common items that may be part of actor state across application domains have also been investigated by the EU Provenance project (<http://www.gridprovenance.org>), which were derived from the GLUE Information Model [6], though due to their application dependence, representation of them has not been specified. Our registry architecture attempts to provide a method whereby actor state assertion capture is formalised, but its content is left customisable.

Often, elements of actor state are captured through use of annotations. In the MyGrid (<http://www.mygrid.org.uk/>) project for instance, provenance data is generated from bioinformatics experiments and classed into: *the derivation path*: by which the results were generated from the input data, and *annotations*: associated with a particular object or collection of objects. Such annotations may include elements of actor state such as version data for workflows and resources [7]. Formal representation of actor state, as well as automation of its collection independent of application constraints, is desirable and would be useful in evaluating the state of actors across research disciplines.

The use of performance data to obtain insights into the relationship between application and hardware and software has previously been explored [8], enabling automatic model generation through performance analysis. Within job-based execution environments, work has been performed to enable provenance recording with a minimal level of system intrusion [9]. Automatic instrumentation of such applications with performance monitoring code is possible due to direct availability of implementation. The trade-offs between the level of intrusion to both the application system and user, necessary to capture adequate provenance information has previously been likened to a cube [9] where intrusion to the system and user are modelled on the x and y axis and the amount of available information on the z-axis. The most desirable system is described as one with no intrusion to system or user, but providing all information about the two. In service oriented systems instrumentation of actors may not be possible, due to their loosely-coupled interaction with the querying actor. The level of intrusion which is possible in such environments is therefore minimal, and as such so is the level of information able to be captured. Our system differs through exploring how service oriented systems (where direct knowledge of implementation may be unknown) may record assertions of actor state using resources which are not necessarily part of the application system, alongside interaction assertions. The combined use of such assertions is possible in two ways, understanding how an actor performed within the context of an interaction and secondly understanding what an actor was doing when a particular performance pattern occurred.

6 Conclusion

The use of assertions of client or service (actor) state are not often documented within service oriented architectures, despite them being a critical component in determining the process which led to a particular data item (its provenance). This has been due to the application dependent nature of such data. Actor state data can be combined with assertions of interaction between actors to enable better reasoning within a provenance system. We have identified a number of requirements for a system capable of actor state capture in an application independent manner and attempted to provide a registry based architecture which is able to satisfy them. As future work, we plan to optimise our prototype to produce a more scalable solution and investigate other sources of monitoring information which could be used within our architecture. Of particular interest is the monitoring of per-actor metrics and patterns of access, especially when multiple actors co-exist on the same platform.

References

1. Groth, P., Jiang, S., Miles, S., Munroe, S., Tan, V., Tsasakou, S., Moreau, L.: An Architecture for Provenance Systems. Technical Report (v0.6), University of Southampton (2006) [Online]. Available: <http://eprints.ecs.soton.ac.uk/12023/>.
2. Miles, S., Groth, P., Branco, M., Moreau, L.: The requirements of recording and using provenance in e-Science experiments. Technical report, University of Southampton (2005) [Online]. Available: <http://eprints.ecs.soton.ac.uk/11189/>.
3. Groth, P., Luck, M., Moreau, L.: A protocol for recording provenance in service-oriented Grids. In: Proceedings of the 8th International Conference on Principles of Distributed Systems (OPODIS'04), Grenoble, France (2004) [Online]. Available: <http://eprints.ecs.soton.ac.uk/11914/>.
4. Ali, A.S., Rana, O.F., Parmee, I.C., Abraham, J., Shackelford, M.: Web-Services Based Modelling/Optimisation for Engineering Design. In: OTM Workshops. (2005) 244–253
5. Kloss, G.K., Schreiber, A.: Provenance Implementation in a Scientific Simulation Environment. In: Proceedings of the International Provenance and Annotation Workshop (IPAW'06), Chicago, USA, Springer-Verlag (2006)
6. Andreozzi, S., Burke, S., Field, L., Fisher, S., Konya, B., Mambelli, M., Schopf, J.M., Viljoen, M., Wilson, A.: Glue Schema Specification. Technical Report (v1.2) (2005) [Online]. Available: <http://infforge.cnaf.infn.it/glueinfomodel/index.php/Spec/V12>.
7. Greenwood, M., Goble, C., Stevens, R., Zhao, J., Addis, M., Marvin, D., Moreau, L., Oinn, T.: Provenance of e-Science Experiments - experience from Bioinformatics. In: Proceedings of the UK OST e-Science second All Hands Meeting 2003 (AHM'03), Nottingham, UK (2003) 223–226
8. Taylor, V.E., Wu, X., Stevens, R.L.: Prophecy: an infrastructure for performance analysis and modeling of parallel and grid applications. SIGMETRICS Performance Evaluation Review **30**(4) (2003) 13–18
9. Reilly, C.F., Naughton, J.F.: Exploring Provenance in a Distributed Job Execution System. In: Proceedings of the International Provenance and Annotation Workshop (IPAW'06), Chicago, USA, Springer-Verlag (2006)

Provenance Collection Support in the Kepler Scientific Workflow System

Ilkay Altintas¹, Oscar Barney², and Efrat Jaeger-Frank¹

¹ San Diego Supercomputer Center, University of California, San Diego,
9500 Gilman Drive, San Diego, CA 92092-0505
{altintas, efrat}@sdsc.edu

² Scientific Computing and Imaging Institute, University of Utah
50 S. Central Campus Drive, Salt Lake City, UT 84112
oscar@sci.utah.edu

Abstract. In many data-driven applications, analysis needs to be performed on scientific information obtained from several sources and generated by computations on distributed resources. Systematic analysis of this scientific information unleashes a growing need for automated data-driven applications that also can keep track of the provenance of the data and processes with little user interaction and overhead. Such data analysis can be facilitated by the recent advancements in scientific workflow systems. A major profit when using scientific workflow systems is the ability to make provenance collection a part of the workflow. Specifically, provenance should include not only the standard data lineage information but also information about the context in which the workflow was used, execution that processed the data, and the evolution of the workflow design. In this paper we describe a complete framework for data and process provenance in the Kepler Scientific Workflow System. We outline the requirements and issues related to data and workflow provenance in a multi-disciplinary workflow system and introduce how generic provenance capture can be facilitated in Kepler's actor-oriented workflow environment. We also describe the usage of the stored provenance information for efficient rerun of scientific workflows.

1 Introduction

Current technology significantly accelerates the scientific problem solving process by allowing scientists to access data remotely, distribute job execution across remote parallel resources, and efficiently manage data. Although an increasing amount of middleware to accomplish these tasks has emerged in the last couple of years, using different middleware technologies and orchestrating them with minimal overhead still remains difficult for scientists. Scientific workflow systems [1,2,3], aim to improve this situation by creating interfaces to a variety of technologies and providing tools with domain-independent customizable graphical user interfaces that combine different Cyberinfrastructure [4] technologies along with efficient methods for using them. Workflows enormously improve data analysis, especially when data is obtained from multiple sources and generated by computations on distributed resources and/or various analysis tools. These advances in systematic analysis of scientific information

made possible by workflows have unleashed a growing need for automated data-driven applications that also collect and manage the provenance of the data and processes with little user interaction and overhead.

Requirements for Efficient Provenance Collection during Scientific Workflow Design and Execution. Scientific problem solving is an evolving process. Scientists start with a set of questions then observe phenomenon, gather data, develop hypotheses, perform tests, negate or modify hypotheses, reiterate the process with various data, and finally come up with a new set of questions, theories, or laws. Most of the time before this process can end in results, scientists will fine-tune the experiments, going through many iterations with different parameters [5]. This repeating process can reveal a lot about the nature of a specific scientific problem and, thus, provides information on the steps leading to the solution and end results. Making this information available requires efficient usage and collection of data and process provenance information. Another very important requirement for any scientific process is the ability to reproduce results and to validate the process that was followed to generate these results. Provenance tracking provides this functionality and also helps the user and publication reviewer/reader understand how the run happened and what parameters and inputs were associated with the workflow run.

A provenance collection system must have the ability to create and maintain associations between workflow inputs, workflow outputs, workflow definitions, and intermediate data products. Collecting intermediate data products in addition to other provenance data serves multiple purposes as the results of some processes in the workflow can vary from run to run and some workflow tests require manually stepping through some of the steps to verify and debug results. The intermediate results along with the other provenance data can be also used to perform “smart” reruns, which will be described in section 5.

These requirements illustrate the strong need to retain origin and derivation information for data and processes in a workflow, to associate results with customized and executable workflow version, and to track workflow evolution as described in [16]. This paper discusses an implementation that aims to keep track of all these aspects of provenance in scientific workflows.

In the rest of this paper, we review the related work and how our contribution differs from the rest of the previous work in this area, give a brief overview of the Kepler scientific workflow system, and introduce a generic provenance collection framework in Kepler. Finally, we explain the “smart” rerun feature that exhibits a usage of extracted provenance information for performing efficient re-runs for a slightly modified workflow in Kepler.

2 Related Work

The need for data provenance has been widely acknowledged and is evident in numerous applications and systems. Here, we give an overview of several research efforts in the field, some of which were also listed in a recent survey of data provenance techniques by Simmhan et al. [6]. We plan to extend the data, execution and workflow provenance capabilities of our provenance framework based on the past work explained below.

The Chimera [7] Virtual Data System uses a process model for tracking provenance in the form of data derivations. Using a Virtual Data Catalog (VDC), defined by a query-able Virtual Data Schema, Chimera represents executables as transformations, a particular execution as a derivation and inputs/outputs as data objects. Workflows in Chimera are defined as derivation graphs through a Virtual Data Language (VDL). The VDL is also used for querying the VDC independent of the catalog schema. Provenance in Chimera is used for tracking the derivation path that led to a data product, reproducibility of a derived product, and validation/verification of an experiment.

In the MyGrid project, provenance data is recorded for workflows in XScufl language that are executed using the Taverna workflow engine [8]. A provenance log is automatically recorded during the workflow execution in a framework differentiating between four provenance levels: The *process level* gathers information about the invoked processes, their inputs/outputs and processing times. The *data level*, inferred from the process level describes intermediate and final products derivation paths. The *organization level* stores the metadata for the experiment, and the *knowledge level* links the experiment's scientific findings/"knowledge" with the other provenance levels as supporting evidence. The stored information is used to infer the provenance of intermediate and final results and for quality verification of the data in terms of tracing the processing steps.

The above "provenance recording systems" are tightly coupled with their workflow execution environment. The Provenance Aware Service Oriented Architecture (PASOA) project aims to provide interoperable means for recording and using provenance data using an open provenance protocol [9]. In [18] PASOA identifies several requirements for a generic, application independent, provenance architecture for e-Science experiments. Among those requirements are recording of *interaction provenance*, *actor provenance* and *input provenance*, where interaction provenance is recording interactions between components and the data passed between them, actor provenance is recording processes information and the time of the execution, and input provenance is tracking the set of input data used to infer a data product. Other requirements include *reproducibility* of an experiment, *preservation* and *accountability* of provenance over time and *customizability* to support and integrate with diverse architectures.

Other applications of data provenance are evident in database and geographic information systems. Data lineage in database systems where data provenance refers to 'a description of the origins of a piece of data and the process by which it arrived in a database' has been addressed by Bunman *et al.* [10, 19]. Bunman *et al.* define the data lineage problem as "why" and "where" provenance. *Why-provenance* refers to the set of tuples that contributed to a data item, where as *where-provenance* defines how a data item is being identified in the source data. In Geographic Information Systems (GIS) data lineage is used for dataset validation. Metadata is recorded for tracking the transformations applied to derive a data item [11].

The VisTrails system [15] was developed to facilitate interactive multiple-view visualizations by providing a general infrastructure, which can be used in conjunction with any existing visualization system, like Kitware's Visualization Toolkit [17], to create and maintain visualization workflows as well as to optimize their execution. Often, progress is made by comparing visualizations that are created by the same

basic workflow, but with slightly different parameters or components. Thus, the Vistrails system collects and maintains a detailed provenance record for each instance of a workflow as well as across different versions of a workflow thus tracking the evolution of the workflow. The Vistrails system is the first system to track a workflow's evolution [16], something that can be useful for anyone who wants to execute a workflow multiple times, store the results, and then compare multiple versions of the workflow in an organized fashion in order to find just the right set of components and parameters.

Provenance tracking is important for scientific computing. This paper discusses an implementation for the Kepler scientific workflow system, which aims to keep track of all aspects of provenance in scientific workflows: in workflow evolution, data and process provenance, and efficient management and usage of collected data. While there are similarities between aspects of the above-mentioned previous work and our own, to the best of our knowledge, this approach to designing a provenance collection framework that is highly configurable, comprehensive, model of computation independent, and includes facility for smart reruns is a unique contribution to provenance research in the scientific workflow community.

3 Kepler Scientific Workflow System

A scientific workflow is the automated process that combines data and processes in a structured set of steps to implement computational solutions to a scientific problem. Kepler [1] is a cross-project collaboration to develop a scientific workflow system for multiple disciplines that provides a workflow environment in which scientists can design and execute workflows.

Kepler builds on top of the mature Ptolemy II software [12], which is a Java-based system and a set of APIs for heterogeneous hierarchical modeling. The focus of Ptolemy II is to build models based on the composition of existing components, which are called 'Actors', and observe the behavior of these simulation models when executed using different computational semantics, which are implemented as components called 'Directors.'

Actors are the encapsulations of parameterized actions performed on input data to produce output data. Actors communicate between themselves by sending Tokens, which encapsulate data or messages, to other actors through ports. An actor can have multiple ports and can only send Tokens to an actor that it is connected to one of its output ports. The director specifies the model of computation under which the workflow will run. For example, in a workflow with a Process Network (PN) director, actors can be thought of as separate threads that asynchronously consume inputs and produce outputs. Under the Synchronous Dataflow (SDF) director, actors share a common thread, and the order of execution is statically determined because the number of tokens each actor will produce and consume is known ahead of time. Also, different domains control how the ports relay Tokens. For example, in PN each port behaves like a FIFO queue of unlimited size where as a port controlled by the SDF director acts like a FIFO queue with a size limited to the number of tokens an actor can produce or consume.

Kepler System Architecture

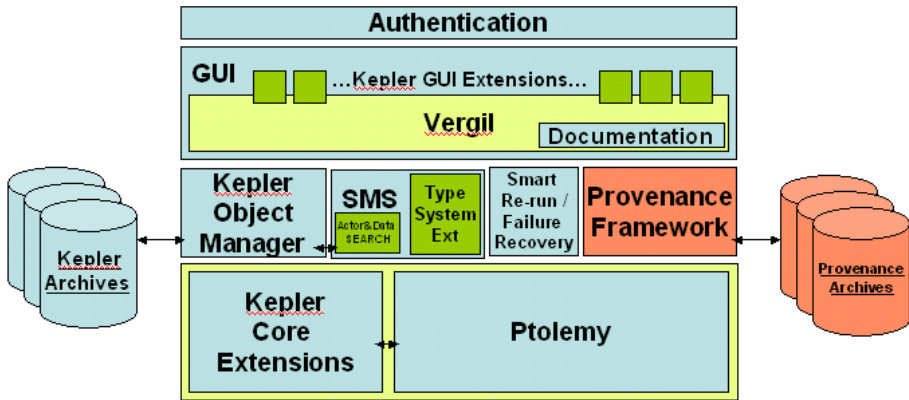


Fig. 1. The Kepler System Architecture

Kepler actors perform operations including data access, process execution, visualization, and domain specific functions. Kepler uses Ptolemy II's hierarchical actor oriented modeling paradigm to create workflows, where each step is performing some action on a piece of data. Workflows can be organized visually into sub-workflows. Each sub-workflow encapsulates a set of executable steps that conceptually represents a separate unit of work. The Kepler system can support different types of workflows ranging from local analytical pipelines to distributed, high-performance and high-throughput applications, which can be data- and compute-intensive. [13] Along with the workflow design and execution features, Kepler has ongoing research on a number of built-in system functionalities, as illustrated in Figure 1, including support for single sign-in GSI-based authentication and authorization; semantic annotation of actors, types, and workflows; creating, publishing, and loading plug-ins as archives using the Vergil user interface; and documenting entities of different granularities on-the-fly.

Ptolemy II separates the description of important aspects of a design such as behavior and architecture, or computation and communication. [14] Kepler inherits this concept of *separation of concerns* in design from the Ptolemy II. This provides significant advantages such as lower design time and better re-usability of the design because system designers can build a new component for the system and plug them in for testing without changing any of the underlying architecture. Also, workflow designers do not have to use ad hoc techniques to implement the workflow's design and execution of the workflow graph. The Ptolemy II system provides a general strategy for separating the workflow composition from the overall orchestration of the model by introducing the separate concerns for actors, their composition, and the implementation of computational domains that run the workflows. These 'separate concerns' are combined visually into a model on the screen, which provides an easy way for

system users to see what the exact behavior of the workflow will be without clicking on menu to find out things like what the model of computation will be, for example.

4 Generic Provenance Framework in Kepler

Given the collaborative and domain-independent nature of the Kepler project, our provenance framework needed to include plug-in interfaces for new data models, metadata formats and cache destinations. To accomplish this goal we have created a highly configurable provenance component that can be easily used with different models of computation using the separation of concerns design principle. Just like a director, provenance collection is modeled as a separate concern that is bound visually to the associated workflow. This way a user can easily see if provenance is being collected for a certain run of the workflow. Another advantage to this design is its compliance with Kepler's visual actor-oriented programming paradigm, and that it is consistent with the behavior of the Kepler user interface.

4.1 Design Objectives

A major objective when designing the provenance recording functionality was ease of use. We did not want the user to have to go through a complex configuration process or the actor designers to have to implement a complex API. Since Kepler is built on top of the Ptolemy II framework, we had to consider designs that would seamlessly integrate with existing code and work with any director.

When designing the provenance collection system, another major consideration was supporting the multi-disciplinary and multi-project nature of the Kepler project. To be more flexible we made our collection facility parametric and customizable. For example, a user may want to limit the granularity of the collected data, publish it in a specific data source, or only save certain results to verify the behavior of a specific workflow component during testing. To facilitate this, we allow the user to choose between various levels of detail, and even save all of the provenance data needed to recreate a workflow result when the workflow is used as a part of the scientific discovery.

A workflow run consists of several pieces of information that need to be recorded including the *context*, *the input data and its associated metadata*, *the workflow outputs*, *intermediate data products*, *the workflow definition*, and information about the *workflow evolution*. Context is the *who, what, where, when, and why* that is associated with the run. Workflow definition is a specification of what exists in the workflow and can have a context of its own. It includes information about the workflow's entities, their parameters and the connections between the actors. Workflow evolution, also known as a workflow trail [16], is a description of how the workflow definition has changed over time. This is an application of the ideas in [16]. By tracking the evolution of a workflow design, its runs, and its parameters over time, the scientist can efficiently manage the search of a parameter space and easily jump back to a previous version of the workflow that produced interesting results.

One of our side goals when designing the provenance recorder was its ability to help us debug a workflow during the implementation phase of workflow development. By mining and analyzing ‘process provenance,’ data related to the execution of the workflow, and intermediate data products that were processed at the time of an error, we may be able to figure out exactly what was happening at the time of an error in our prototype workflow.

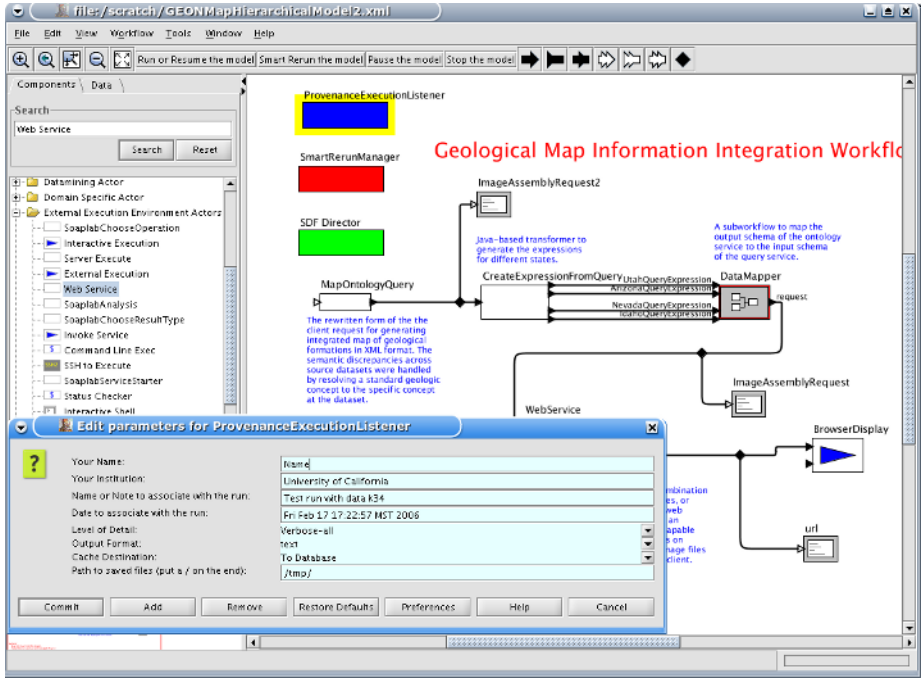


Fig. 2. A screenshot of Vergil that shows the different concerns for model of computation (green), provenance collection (blue), smart rerun (red) and actors

4.2 Implementation

To address ease of use, we designed the *Provenance Recorder (PR)* to be configured and represented in the same way as a Director in Kepler. To enable provenance collection in a workflow instance the user drags the PR from the toolbox and places it on the workspace with the Director and other workflow entities. Unlike using Directors, using the PR with a given workflow is optional, depending on the user’s requirements for tracking provenance. Similar to the Director, it is configured with a standard configuration menu and becomes part of the workflow definition. (See Fig. 2.)

We had to pursue the following steps to provide automatic collection of provenance information in Kepler. We converted Kepler’s internal XML workflow representation, MOML, into our internal format for provenance data. This format leaves out some of the unnecessary information MOML includes (i.e. actor coordinates and the custom actor icons in the user interface) and includes extra information (i.e. Token

production rate of the actors, etc.) critical to complete data provenance collection. Using the existing MOML generation capabilities in Kepler helped us to efficiently collect the provenance associated with the workflow definition. Also, by keeping track of all information associated with the workflow definition we are able to track workflow evolution and jump back to interesting workflows saved in our provenance store. We leave out the details of our internal format for provenance collection and caching here as it is out of the scope of this paper.

In order to collect information that is generated during a workflow run, the PR implements several event listener interfaces. Different ‘concerns’ in Ptolemy II and Kepler, such as the ports through which actors communicate, maintain a list of event listener objects that are registered with them. When something interesting happens, the event listeners registered with the specific ‘concern’ in question are notified, and take the appropriate action. For example, when the PR is notified that a data product is created, it can then associate the appropriate data lineage information with this data product and put it in the provenance store. Event listeners are also allowed to register and un-register with individual concerns so that we can easily control the amount of provenance data that is collected during any one run. This can be very important because some workflows create a massive number of intermediate data products, which are not always necessary to recreate the results of a certain workflow.

When the workflow is loaded, the PR will register with the appropriate ‘concerns’ in the workflow. When the workflow is executed, PR will process information received as events, and save it in provenance store. As we have mentioned before, the provenance recorder can save information that is useful for debugging the workflow. To accomplish this, we have the PR register with the appropriate concerns that send out notification of events related to the execution of the workflow and any errors that occur. In this way we can find out exactly what actor was executing, with what inputs when a certain error occurred.

Although we were able to automate much of the provenance collection, we had to extend the design to handle several other cases. For example, if an actor creates an external data product, it must register this product with the PR as well as reporting its internal actions. We developed a simple API allowing actors to notify the PR in these situations. Once the actors are extended using this API, the PR can collect and save these data products and actions in addition to any local data products that were automatically collected using the event listener interfaces.

5 Efficient Workflow Rerun Enabled by Provenance Data

In Kepler, we have added functionality to enable efficient reruns of a workflow by mining stored provenance data. The idea behind a “smart” rerun [1] is as follows. When a user changes a parameter of an actor then runs the workflow again, re-executing all the preceding, unchanged steps (actors) in the workflow may be redundant and time consuming. A “smart” rerun of the workflow will take data dependencies into account and only execute those parts of the workflow affected by the parameter change. The ability to store and mine provenance data is required to enable “smart” reruns since the intermediate data products generated in previous runs are used as the inputs to the actors that are about to be rerun.

We have created the Smart Rerun Manager (SRM) to handle all the tasks associated with the efficient rerun of a workflow. This includes data dependency analysis and provenance data management. The algorithm used by the SRM is derived from the Vistrail execution and cache management algorithm used in the Vistrails system [15]. The VisTrails cache management algorithm was developed to allow users of a visualization system to efficiently explore a parameter space. The premise is that we can extract intermediate results of the dataflow from a cache instead of recreating them in order to save time when rerunning the dataflow.

5.1 Implementation and Algorithm Description

The SRM is an event-based entity in the workflow, which is a 'separate concern' in the Kepler system just like the PR. They are both used and configured in a similar way. Once a SRM is placed on the workspace by dragging it from the toolbox, all provenance data needed to perform a smart rerun of the workflow will be collected. To allow users to choose whether or not they want to perform a smart rerun of the workflow, the SRM is activated by pressing a special "Smart Rerun" button in the user interface next to the standard "Run Workflow" button. In this section, we describe how the SRM system uses provenance data for optimized rerun and our changes to the underlying Vistrails algorithm, which the SRM builds upon.

The basic idea behind the Vistrails algorithm is to search a graph representation of the dataflow for sub-graphs that can be eliminated. The precondition for elimination of these sub-graphs is that the actors they contain have already been run with the current parameters and input data. The next step is to retrieve the intermediate data products produced by this eliminated sub-graph from the provenance store for use as input to the actors that need to be rerun. This part of the provenance store we will call the provenance cache. Each sub-graph is identified with a unique ID, which is an important concept that we borrow from the Vistrails system. A unique ID, which is used as a key when searching the provenance cache for information related to a particular sub-graph, represents a unique state of a component (a.k.a. actor), its parameters, and all the actor and parameters that come before it in the workflow. Each unique ID is associated with a specific actor and encapsulates the provenance information needed to uniquely identify and retrieve the intermediate data products produced by that actor.

When activated by pressing the special "Smart Rerun" button in Kepler's toolbar, the SRM builds a directed graph representing the data dependencies in the workflow. Each node in the graph represents an actor in the workflow and the edges represent the flow of data between actors. The SRM then analyzes this graph to detect sub-graphs that have been successfully computed before and the sub-graphs that must be rerun.

The analysis begins at the graph's sinks and recursively traverses all the input edges of each node in the direction opposite to the flow of data. At each node, a unique ID is generated and used as a key in the cache lookup. If this unique ID is associated with some data in the provenance cache, this means that the workflow as it exists from this node backward in the dependency graph has been executed successfully before. In this case the actors represented by the nodes sub-graph corresponding to the unique ID can be eliminated from the list of actors to be rerun. Conceptually, the intermediate data product retrieved from the provenance store using the unique ID replaces this sub-graph.

How this is done in practice will be clarified soon. If this unique ID is not associated with any data products in the provenance store, we keep traversing our graph until we do find a unique ID with associated data in the provenance cache or we reach the source nodes in the graph and they need to be rerun as well.

The last step of the “smart” rerun process is to replace the eliminated sub-graphs with components that can stream the data from the provenance cache. In Kepler-specific terms, we need to place the intermediate data products retrieved from the provenance cache on the appropriate actors’ input ports while the workflow is running. These intermediate data products are the tokens that flowed across this input edge from the eliminated actors in the previous runs of the workflow. We developed a special actor to replace the eliminated actors and replay the tokens that they would have produced. This special actor is called the *Stream Actor* because it streams data from the provenance store into the running workflow. This piece of the system is called the *vtkDataObjectPipe* in Vistrails. Figure 3 visually illustrates a simple example where the SRM retrieves the intermediate results of the previous execution and replaces the previously executed parts of the workflow with a streaming actor.

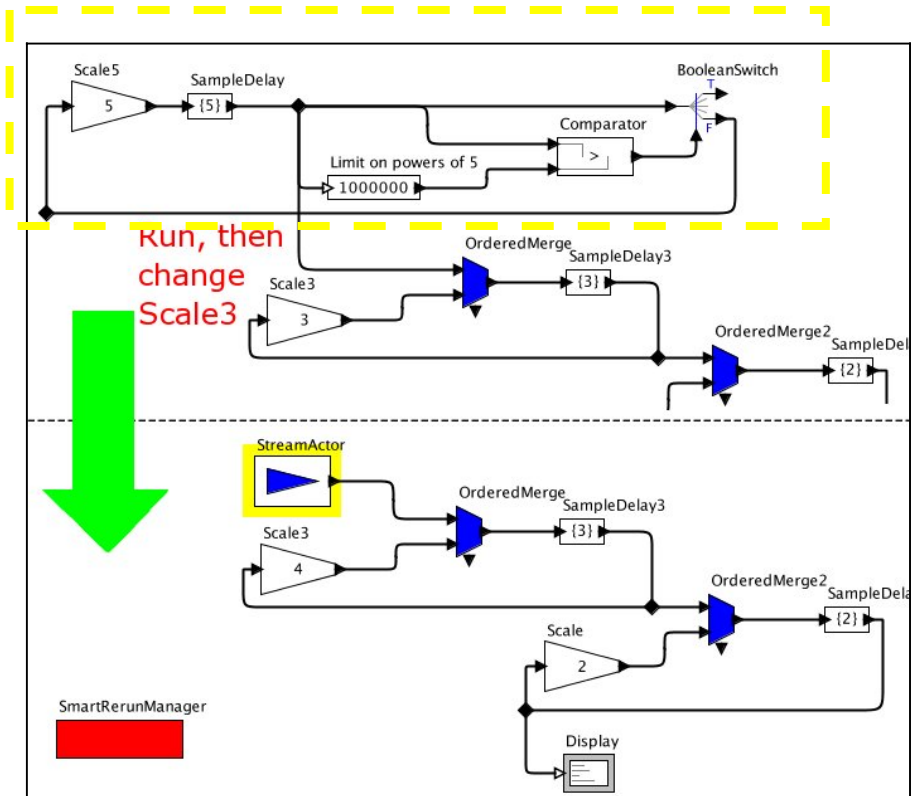


Fig. 3. Smart Rerun Manager retrieving the intermediate results of the previous execution and replacing the pre-executed parts of the workflow with a streaming actor. Bottom workflow shows *StreamActor* replacing actors whose computations would be redundant in the rerun.

It is important to note that the data associated with a successful lookup in the provenance cache can be associated with the preceding run of the workflow or any previous runs with the same components, connections, and parameters thus utilizing all past provenance information. Also, two sub-graphs of the same workflow could potentially be replaced by data from two distinct runs of the workflow. This is a major strength of the Vistrails algorithm, which ensures that no work is repeated.

For the most part, the SRM is able to use the Vistrails cache management algorithm, however we have made some noteworthy changes and additions. The Vistrails system was designed with a single model of computation in mind and actually executes portions of the workflow as a part of its graph analysis stage. Since the SRM must distinguish between different models of computation (Directors) in Kepler that expect different behaviors during the workflow run, we had to do the graph analysis step before handing off the execution of the workflow to the Director. This is an efficient design in Kepler because each Director has a distinct behavior and trying to encode these behaviors in the SRM would result in redundant code.

Also, our Stream Actor must take these separate models of computation into account. For example, when a workflow is executed using a Director, such as the Process Networks Director that requires each actor consume a stream of inputs and produce a stream of outputs, the SRM actor makes sure that the stream of tokens is replayed in the same order that it was collected when inserting data from the provenance store into the workflow. In a domain, such as Synchronous Dataflow (SDF), where each actor consumes a certain number of inputs and produces a certain number of outputs, the Stream Actor gives the tokens to input ports at the rate they are expected. What we mean by rate is clarified by the following example. In a model controlled by the SDF Director assume that actor A declares that it will consume x tokens each time it is activated. The SDF Director schedules the actors so that they will not run until they have the proper number of inputs. If the actor B creates $x/2$ inputs each time it executes and is connected to actor A, the SDF Director will schedule B to execute twice so that the actor A will have enough inputs when it executes. As it is illustrated in this example, the SRM must guarantee the production rate of the StreamActor to ensure that the rerun is performed correctly.

Another difference between the Vistrails algorithm and the algorithm that the SRM is using, is the way in which the SRM handles ‘non-cacheable’ actors in the workflow. Non-cacheable actors are the actors whose output depends on when the run occurs as well as what the inputs and parameters are. For example, an actor that queries a remote database is non-cacheable if the database is modifiable because it may receive different results depending when the query is executed. In contrast, the Vistrails system views every component as a function, for a specified input you can predict the output. A non-cacheable actor in the Vistrails system does not have its outputs saved, and thus its unique ID will never be found in the provenance cache. Non-cacheable components in the context of Visualization workflows are those whose outputs cannot be saved or whose outputs are too large to be saved. Actors that depend on the non-cacheable actor are not rerun unless there is a new input or parameter change upstream. If it is not specified otherwise, the SRM will rerun all actors that depend on a non-cacheable actor since their results depend on the non-deterministic nature of the non-cacheable actor.

The SRM's user interface allows the user to specify if an actor is cacheable or if it must be rerun every time. In some cases you may want to save the state of an actor that behaves in a non-deterministic way. This enables the user choose between doing a "smart" rerun of the workflow with saved provenance data to exactly recreate a past run or to rerun the workflow to get the most up-to-date results but still avoid redundant steps.

The SRM is an example to valuable usage of data from our provenance store. It has the potential to save scientists hours while they explore the parameter space of their workflows and is an important feature of the Kepler system.

6 Results and Conclusions

This paper discusses our generic provenance framework for use with scientific workflows. The framework is designed to support a wide range of workflow types and is extensible because of the modularity of its design and the flexibility of the event listener interfaces that it implements. Most of the discussed functionality has been implemented with the exception of a final data model design. This paper does not focus on the internal structure of the collected information to support provenance and caching, but mentions these to explain the PR and SRM. We plan to continue working on the data models and make it available in the near future. We have already had interest in the PR from a wide variety of users, some of which have used our initial version and given positive feedback.

Performance Evaluation. The PR has been designed to be as generic as possible and has met most of the design goals that we set out to achieve. The event based nature of the design has allowed us to collect the variety of information needed in order for the system to be useful in a wide range of application areas while at the same time having a minimal performance impact on the system. Specific performance measurements for workflows using the PR vary greatly depending on the amount of provenance data being saved and the ratio of data produced to time spent computing. For example, a computationally intensive workflow may produce the same amount of provenance data as a workflow that runs in a matter of seconds, but has less overhead as the PR takes much smaller percentage of total run time. We can safely say that we have accomplished our design goal of efficiency because in the majority of our test cases the increase in run time attributed to the PR is minimal and usually only a couple of seconds. Also, in some cases where a specific actor generates excessive amounts of data, our design allows us to specify that we are not interested in this actor's information by un-registering with its list of event listeners.

7 Future Work

We have developed several prototype relational and XML data models and plan to implement them in the Kepler Provenance Framework once we have design a suitable data model. The data model for storing provenance information in Kepler should accommodate the needs of different scientific domains as well as allow for efficient storage and retrieval of data. Another area of we are interested in researching is in

defining the policies for managing provenance data for different projects. This is an important problem that utilizes other functionality and system components in Kepler including the authentication and authorization framework, the data access API, and semantic annotations. Another planned usage of actor annotations is to annotate the actors so that the PR could automatically figure information related to what files they create during the run, what algorithm and data structures they use, etc. We also plan to implement querying and viewing system for collected provenance information.

There are multiple provenance activities within the Kepler collaboration including provenance tracking in collection-oriented workflows and integrating with RDF based provenance stores. In this paper, we have only mentioned the existing implementation, the algorithms it utilizes, and its functionality. We plan to bring the ongoing research by other Kepler developers and researchers together under the Kepler provenance framework once these research ideas are implemented and become available for public use. In particular we would like to experiment with the provenance model developed by Bowers et al. [20] to support a wide range of scientific use cases in phylogeny and the data model for the XMLSchema-based arbitrary textual and binary data format articulation capability by Talbott et al. [21].

This paper already described the usage of the Vistrails algorithm for smart re-runs of the same workflow. We plan to further incorporate the Vistrails system capabilities into our provenance framework for systematically capturing detailed provenance and workflow evolution information. [22] This will require customizing or extending the existing Vistrails action-based model by information on the Kepler workflow modeling language (MoML) and updating the core Kepler modeling components to record this information during the modeling and experimentation phase for a scientific workflow.

Acknowledgements

The authors would like to thank the rest of the Kepler team for their excellent collaboration, especially to Timothy McPhillips for the discussion on requirements for a provenance framework and future steps, Claudio Silva and Juliana Freire for their help on VisTrails and insight on workflow provenance, and Steven Parker for his support and guidance. This work was supported by DOE Sci-DAC DE-FC02-01ER25486 (SDM) and NSF/ITR 0225673 (GEON).

References

1. Ludäscher, B., Altintas, I., Berkley, C., Higgins, D., Jaeger-Frank, E., Jones, M., Lee, E., Tao, J., Zhao, Y.: Scientific Workflow Management and the Kepler System. *Concurrency and Computation: Practice & Experience*, Special Issue on Scientific Workflows, to appear, 2005. <http://kepler-project.org/>
2. Oinn, T., Greenwood, M., Addis, M., Alpdemir, M.N., Ferris, J., Glover, K., Goble, C., Goderis, A., Hull, D., Marvin, D., Li, P., Lord, P., Pocock, M.R., Senger, M., Stevens, R., Wipat, A. Wroe, C.: Taverna: Lessons in creating a workflow environment for the life sciences". Accepted for publication in *Concurrency and Computation: Practice and Experience Grid Workflow Special Issue*

3. Churches, D., Gombas, G., Harrison, A., Maassen, J., Robinson, C., Shields, M., Taylor, I., Wang, I.: Programming Scientific and Distributed Workflow with Triana Services". In Grid Workflow 2004 Special Issue of Concurrency and Computation: Practice and Experience, to be published, 2005
4. Revolutionizing Science and Engineering Through Cyberinfrastructure: Report of the National Science Foundation Blue Ribbon Advisory Panel on Cyberinfrastructure
5. Lipps, J. H.: The Decline of Reason?. <http://www.ucmp.berkeley.edu/fosrec/Lipps.html>,
6. Simmhan, Y. L., Plale, B., Gannon, D., A survey of data provenance in e-science. In *SIGMOD Rec.* 34(3): 31-36, 2005
7. Foster, I., Voeckler, J., Wilde, M., Zhao, Y.: Chimera: A Virtual Data System for Representing, Querying, and Automating Data Derivation. In Proceedings of the 14th Conference on Scientific and Statistical Database Management, 2002
8. Greenwood, M., Goble, C., Stevens, R., Zhao, J., Addis, M., Marvin, D., Moreau, L., Oinn, T.: Provenance of e-Science Experiments - experience from Bioinformatics. In Proceedings of The UK OST e-Science second All Hands Meeting 2003 (AHM'03)
9. Groth, P., Luck, M., Moreau, L.: A protocol for recording provenance in service-oriented grids. In Proceedings of the 8th International Conference on Principles of Distributed Systems (OPODIS'04), 2004
10. Buneman, P., Khanna, S., Tan, W.C.: Why and Where: a characterization of data provenance. In Proc. ICDT 2001
11. Lanter, D.P., Design of a lineage-based meta-data base for GIS, In *Cartography and Geographic Information Systems*, 18(4):255-261, 1991
12. Ptolemy Project, See Website: <http://ptolemy.eecs.berkeley.edu/ptolemyII/>
13. Altintas, I., Birnbaum, A., Baldrige, K.K., Sudholt, W., Miller, M., Amoreira, C., Potier, Y., Ludaescher, B.: A Framework for the Design and Reuse of Grid Workflows. Lecture Notes in Computer Science, Scientific Applications of Grid Computing: First International Workshop, SAG 2004, Beijing, China, September 20-24, 2004, Volume 3458 (3), pp 119-132, ISBN3-540-25810-8.
14. Yang, G., Watanabe, Y., Balarin, F., Sangiovanni-Vincentelli, A.: Separation of Concerns: Overhead in Modeling and Efficient Simulation Techniques. Fourth ACM International Conference on Embedded Software (EMSOFT'04), September, 2004
15. Bavoil, L., Callahan, S., Crossno, P., Freire, J., Scheidegger, C., Silva, C., and Vo, H.: Vis-trails: Enabling interactive multiperview visualizations. In *IEEE Visualization 2005*, pages 135-142, 2005
16. Callahan, S., Freire, J., Santos, E., Scheidegger, C., Silva, C., and Vo, H.: Managing the Evolution of Dataflows with VisTrails. In Proceedings of the IEEE Workshop on Workflow and Data Flow for Scientific Applications (SciFlow 2006)
17. The Visualization Toolkit (VTK), See Website: <http://public.kitware.com/VTK/>
18. Miles, S., Groth, P., Branco, M., Moreau, L.: The requirements of recording and using provenance in e-Science experiments. Technical Report, Electronics and Computer Science, University of Southampton, 2005
19. Buneman, P., Khanna, S., Tan, W. C.: Data Provenance: Some Basic Issues. In Proceedings of the 20th Conference on Foundations of Software Technology and theoretical Computer Science, 2000
20. Bowers, S., McPhillips, T., Ludaescher, B., Cohen, S., Davidson, S.B.: A Model for User-Oriented Data Provenance in Pipelined Scientific Workflows . In Proceedings of the IPAW'06 International Provenance and Annotation Workshop, Chicago, Illinois, USA May 3-5, 2006

21. Freire, J, Silva, C.T., Callahan, S.P., Santos, E., Scheidegger, C.E, Vo, H.T.: Managing Rapidly-Evolving Scientific Workflows. In Proceedings of the IPAW'06 International Provenance and Annotation Workshop, Chicago, Illinois, USA May 3-5, 2006
22. Talbott, T.D., Schuchardt, K.L., Stephan, E.G., Myers, J.D.: Mapping Physical Formats to Logical Models to Extract Data and Metadata: The Defuddle Parsing Engine. In Proceedings of the IPAW'06 International Provenance and Annotation Workshop, Chicago, Illinois, USA May 3-5, 2006

A Model for User-Oriented Data Provenance in Pipelined Scientific Workflows*

Shawn Bowers¹, Timothy McPhillips¹, Bertram Ludäscher^{1,2},
Shirley Cohen³, and Susan B. Davidson³

¹ UC Davis Genome Center, University of California, Davis

² Department of Computer Science, University of California, Davis

³ Computer and Information Science, University of Pennsylvania

{sbowers, ludaesch, tmcphillips}@ucdavis.edu

{shirleyc, susan}@cis.upenn.edu

Abstract. Integrated provenance support promises to be a chief advantage of scientific workflow systems over script-based alternatives. While it is often recognized that information gathered during scientific workflow execution can be used automatically to increase fault tolerance (via checkpointing) and to optimize performance (by reusing intermediate data products in future runs), it is perhaps more significant that provenance information may also be used by scientists to reproduce results from earlier runs, to explain unexpected results, and to prepare results for publication. Current workflow systems offer little or no direct support for these “scientist-oriented” queries of provenance information. Indeed the use of advanced execution models in scientific workflows (*e.g.*, process networks, which exhibit pipeline parallelism over streaming data) and failure to record certain fundamental events such as *state resets* of processes, can render existing provenance schemas useless for scientific applications of provenance. We develop a simple provenance model that is capable of supporting a wide range of scientific use cases even for complex models of computation such as process networks. Our approach reduces these use cases to database queries over event logs, and is capable of reconstructing complete data and invocation dependency graphs for a workflow run.

1 Introduction

The importance of provenance information in scientific data and workflow management is widely recognized, as witnessed, *e.g.*, by specialized workshops [4,1], research projects [17], and surveys [3,20] dedicated to this topic, and by investigations on foundations of data provenance for queries and transformations [5,9,23]. However, current scientific workflow systems still offer little or no support for queries of interest to the end-users of these systems, *e.g.*, researchers in

* Work supported in part by SciDAC/SDM (DE-FC02-01ER25486), NSF/SEEK (DBI-0533368), and NSF/GEON (EAR-0225673).

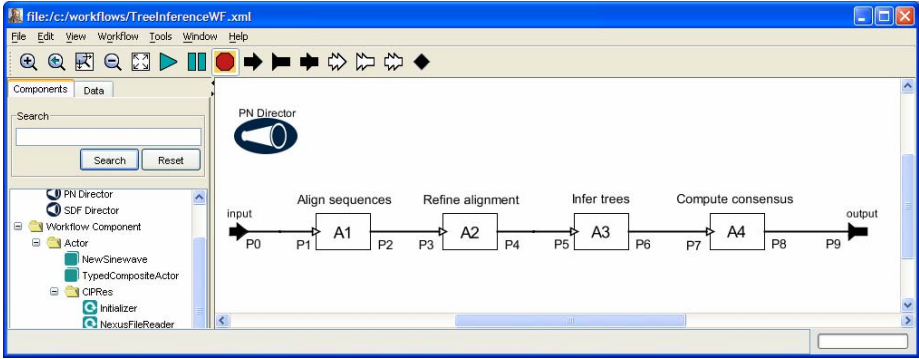


Fig. 1. A workflow for computing phylogenetic trees from input DNA sequences

the life or physical sciences. In this paper, we argue that concrete use cases, expressed in terms that are meaningful to the scientist, should drive the design of a provenance system. Moreover, such systems should be designed in terms of the models of computation (MoC) that govern the execution of scientific workflows to ensure that all pertinent events are recorded in the execution log.

Fig. 1 shows an example workflow for inferring phylogenetic trees approximating the evolutionary relationships between organisms. DNA sequences for homologous genes from a number of taxa are provided as input to the workflow. Actor A1 performs an initial alignment of the sequences (*e.g.*, using the program *ClustalW* [22]), and actor A2 refines this initial alignment (*e.g.*, using *Gblocks* [7]).¹ Actor A3 infers a set of phylogenetic trees from the aligned sequences (*e.g.*, using *DNAPARS* [19]), and actor A4 computes the consensus of these trees (*e.g.*, using *CONSENSE* [19]).

For such scientific workflows we would like to: (a) enable scientists to ask “scientific” questions about a workflow run by providing convenient queries against the run’s execution log; and (b) have the system track the true data dependencies within a run so that answers to such scientific questions may be as accurate as possible. For example, the system should recognize *independent* “sub-runs” as such: The workflow in Fig. 1 may process multiple sets s_1, s_2, \dots of independent DNA sequences (*e.g.*, corresponding to distinct genes in the taxa of interest) within a single workflow run R . In such cases, the system should *not* infer that the data products resulting from the different s_i are interdependent. Rather, the system should answer accurately questions such as:

- Which phylogenetic trees were used to produce this consensus tree?
- Which DNA sequences does this consensus tree depend on?
- Which of the input DNA sequences were **not** used to derive any output consensus tree?

¹ Following KEPLER terminology, we call workflow components *actors*.

In this paper, we develop a provenance model designed to support such user-oriented queries for pipelined models of computation where tracking data dependencies can be complex. For another example, consider an actor A in an environmental monitoring workflow that computes a running average of temperature for each received measurement data token. Thus, upon each invocation or *firing* of A , the actor consumes a temperature token and emits a new running average token. To calculate the running average over multiple firings, A must maintain *state*. For the provenance system this means that every produced data token must be recorded as dependent not just on those input tokens received since the last time the actor fired, but on all tokens received since A was initialized. Conversely, if A is to limit the running averages to readings taken on a particular day, then A 's state is *reset* once per day. There are *no* dependencies between tokens produced after a reset and tokens consumed prior to the reset. This observation naturally partitions token streams, as well as actor firings, into semantically meaningful firing *rounds*.² Clearly, a provenance system should be able to observe and record new rounds of firing to avoid reporting false dependencies.

The running average example described above illustrates a general property of scientific workflows implemented as process networks [12,15,13]: actors need not produce output tokens derived exclusively from tokens received since the last output token was produced. That is, actors in process networks do not generally compute functions on sets of consecutively received inputs. Rather, they may carry out arbitrarily complex transactions on streams of inputs, including running averages, filters, sliding windows, and iterative computations.

Capturing these transaction boundaries is essential for accurately recording scientific workflow provenance. In this paper, we show how this essential information can be represented in a simple tabular event log. Our approach is easy to implement, *e.g.*, in the KEPLER scientific workflow system, where token-read and token-write events can be automatically captured by the workflow framework. Announcing a new round of firing (*e.g.*, by signaling a reset event), on the other hand, is performed by actors themselves, which “know” when they are beginning an independent task (such as a “sub-run” s_i above, or new daily average temperatures).

The rest of this paper is organized as follows. Section 2 briefly overviews scientific workflows within KEPLER, focussing on pipelined execution models. Section 3 presents our provenance model, which consists of read, write, and state reset events. We also describe in Section 3 how to compute data and actor dependency graphs (*e.g.*, for computing data lineage) from corresponding event logs. Section 4 describes a set of operations (or views) over the provenance model for supporting “scientist-oriented” provenance queries. A number of examples are given, which define parameterized queries for the workflow of Figure 1. Finally, Section 5 summarizes our contributions and future work.

² A round is somewhat analogous to a database transaction, specifically in that it constitutes a logical unit of work.

2 Preliminaries

2.1 Workflow Graphs, Actors, and Tokens vs. Data Objects

We adopt notions and terminology from KEPLER, a scientific workflow system extending PTOLEMY II. Workflows are composed by placing *actors* on a design canvas, and “wiring” them together to form the desired workflow graph (Fig. 1). Actors communicate through their input and output *ports*. In a *workflow graph* W , output ports can be connected to input ports, establishing unidirectional dataflow *channels*. Actors communicate through these channels by passing *tokens*.

By default tokens are immutable and “disposable”, *i.e.*, every token t is written only once [15] and thus lives only between its creation on an output port, and its consumption at subsequent input ports. Thus, even if an actor passes on a data object unchanged, a new token-id is created, facilitating tracking of token dependencies. A separate object-id is used to track object dependencies. By $object(t)$ we denote the data *object* represented by the token t . To support user-oriented queries, we associate with an object o one or more types $types(o)$.

The *ports* of an actor A are denoted $ports(A)$. We assume that port-ids are globally unique, *i.e.*, they include a unique actor-occurrence-id and a port-name which is unique to the actor occurrence. A port is either an *input* or *output*, so $ports(A) = in(A) \cup out(A)$. Some input ports $pars(A) \subseteq in(A)$ may be distinguished as *parameters* for configuring A 's behavior. The signature $\Sigma_W := in(W) \rightarrow out(W)$ of a workflow W is given by a set of distinguished inputs $in(W)$ and outputs $out(W)$. As shown in Figure 1, the distinguished workflow input and output ports are connected to a subset of the input and output ports of the workflow's actors.

2.2 Directors

The model of computation (MoC) of a workflow is *not* defined by actors, but specified by a separate component called a *director*. Thus, KEPLER allows workflow designers to choose among different MoCs by choosing appropriate directors. A director specifies and (effectively) mediates all inter-actor communication, separating workflow scheduling and runtime orchestration (a director's concern) from individual actor execution (an actor's concern). This separation achieves a form of *behavioral polymorphism* [14], resulting in more reusable actor components. KEPLER provides a variety of directors that implement process network (PN and SDF), discrete event (DE), continuous time (CT), and finite state transducer (FST) semantics.

2.3 Pipelined Execution

In the process network MoC, the PN director executes each actor as a separate process (or thread). Channels are used to send and to buffer token streams

between actors. Each actor can decide independently how many tokens to consume before writing out a number of output tokens. In this way, workflows that run using the PN director not only exhibit task parallelism, but also *pipeline parallelism*. For example, during a single workflow run, each actor in Figure 1 can execute multiple times, and different actors can execute concurrently.

A number of other MoCs can be considered as special cases of the basic process network model [12,15]. In the synchronous dataflow (SDF) model [13], actors *a priori* define fixed token consumption and production rates. This model allows the SDF director to statically schedule actors, while guaranteeing, *e.g.*, that (unlike in the general PN case) deadlocks cannot occur and that buffers have a fixed size. By DAG (directed acyclic graph) we denote a MoC that is common in job-centric grid workflows [21,10]: nodes represent jobs, and directed edges represent execution dependencies between jobs. Thus, a DAG director can simply execute the jobs in the partial order implied by the job dependency graph. This can be seen as a limited special case of SDF, with an acyclic workflow graph, actors having at most one input and one output port, consuming and producing a single token per workflow run, respectively, and in which each actor is invoked exactly once (unlike in the more general SDF or PN cases).

3 A Provenance Model for Pipelined Workflows

In this section we describe a provenance model that can handle the process network (PN) model of computation, and thus specialized versions such as SDF and DAG as well. To execute a *workflow* (graph) \mathcal{W} , we must “bind” (*i.e.*, select) *input data* \mathbf{i} on which \mathcal{W} will operate. Often \mathcal{W} is also parameterized using initial parameter settings \mathbf{p} . It is customary to record identifiers for \mathcal{W} , \mathbf{p} , and \mathbf{i} as part of the provenance information. Finally, a MoC M is needed (*e.g.*, PN, SDF, DAG) to determine how the workflow is executed.³ Taken together, the equation

$$\mathbf{o} = M(\mathcal{W}_{\mathbf{p}}(\mathbf{i}))$$

denotes a workflow execution in which the output \mathbf{o} is obtained by applying a suitable model of computation M to an appropriately instantiated workflow \mathcal{W} .

3.1 Runs, Traces, and Observables

Each MoC M formally defines the notion of legal *computations* or *runs*, such that one can determine whether a particular run R of a workflow \mathcal{W} is a legal representation (*w.r.t.* M) of an execution $\mathbf{o} = M(\mathcal{W}_{\mathbf{p}}(\mathbf{i}))$. A workflow *trace* \mathcal{T} is an approximation of a run R , according to a *model of provenance*. As recorded by a provenance model, a trace approximates a run by recording functional and non-functional *observables*. For example, an SDF director precomputes a static workflow schedule (based on actor consumption and production rates),

³ Some MoCs might also be aware of resources such as cluster (or grid) nodes and transport protocols, and schedule a distributed workflow accordingly.

and using this schedule signals each actor to fire in turn. Thus, actor firings are directly observed in SDF. In contrast, the *size* of a token (or rather the object it represents) and the *timestamp* when the token was created are *non-functional* observables: according to the MoC, the outcome does not depend on these. Non-functional observables can be useful to record, *e.g.*, to benchmark actor execution times or data transfer times between actors, but are not essential for determining data dependencies.⁴

3.2 The Read, Write, State-Reset (RWS) Provenance Model

Here we consider a concrete model of provenance, called the RWS model, which records *read*, *write*, and *state-reset* events for each actor in a workflow run. These events are stored in a relational *event log*. This model focuses on only a minimal set of observables that allow us to answer many science-oriented user questions (see next section), while ignoring non-functional observables such as timestamps, although such information can be easily added. Figure 2 is an example of an event log for a run of the workflow given in Figure 1. During a workflow run, a read event is added to the event log each time an actor reads a token from a port. Similarly, a write event is added to the log each time an actor writes a token to a port. A series of reads followed by writes denotes an actor *firing*. Note that in a particular firing F_j , an actor may use data that it read in a previous firing F_i to generate output (*e.g.*, this is the typical behavior of a running-average actor, as described in Section 1). In this case, we say the actor maintains state across firings, and state-reset events denote when the state is “flushed” (reset). The firings between reset events constitute a firing round.⁵

As shown in Figure 2, each row in an event log contains: the *location* E_{loc} of the event, which is either a port (for read and write events) or an actor (for state-reset events); the *event type* E_{typ} , which is either ‘r’ for read events, ‘w’ for write events, or ‘s’ for state-reset events; the *token identifier* E_{tok} that was read or written at the port (null for state-reset events); and a *firing count* E_{fire} .

Because actor port identifiers are unique across a workflow, and tokens are written once, the port and token identifiers recorded for each read and write event enable the reconstruction of the flow of data through the workflow run. However, these events alone are not sufficient to reconstruct data dependencies. We use the state-reset events (as described above) along with the firing count for this purpose. In particular, the firing count is incremented independently for each actor whenever (1) an actor switches from writing tokens to reading tokens, denoting a new firing of the actor, and (2) whenever a state-reset event occurs.

⁴ In existing systems, such timestamps are often the only information available and thus are also used to second-guess other properties such as token and object dependencies.

⁵ We use a single state-reset event as opposed to separate events for marking the start and end of a transaction.

E_{loc}	E_{typ}	E_{tok}	E_{fire}	E_{loc}	E_{typ}	E_{tok}	E_{fire}	E_{loc}	E_{typ}	E_{tok}	E_{fire}	E_{loc}	E_{typ}	E_{tok}	E_{fire}
p_0	w	t_1	1	A_1	s	-	3	A_2	s	-	4	p_7	r	t_{25}	1
...				p_1	r	t_{17}	3	A_3	s	-	1	p_7	r	t_{26}	1
p_0	w	t_{18}	1	p_1	r	t_{18}	3	p_5	r	t_{22}	1	p_8	w	t_{29}	1
A_1	s	-	1	p_2	w	t_{21}	3	p_6	w	t_{24}	1	A_4	s	-	2
p_1	r	t_1	1	A_1	s	-	4	p_6	w	t_{25}	1	p_7	r	t_{27}	2
...				A_2	s	-	1	p_6	w	t_{26}	1	p_7	r	t_{28}	2
p_1	r	t_7	1	p_3	r	t_{19}	1	A_3	s	-	2	p_8	w	t_{30}	2
p_2	w	t_{19}	1	p_4	w	t_{22}	1	p_5	r	t_{23}	2	A_4	s	-	3
A_1	s	-	2	A_2	s	-	2	p_6	w	t_{27}	2	p_9	r	t_{29}	1
p_1	r	t_8	2	p_3	r	t_{20}	2	p_6	w	t_{28}	2	p_9	r	t_{30}	1
...				p_4	w	t_{23}	2	A_3	s	-	3				
p_1	r	t_{16}	2	A_2	s	-	3	A_4	s	-	1				
p_2	w	t_{20}	2	p_3	r	t_{21}	3	p_7	r	t_{24}	1				

Fig. 2. The event log for a run of the example workflow in Fig. 1

3.3 Complex Workflow Transactions

In the example event log of Figure 2, state-reset events denote “sub-runs”, *i.e.*, independent actor firings operating on sets of associated data. Note that for the particular event log shown, state reset events occur exactly at read/write transitions (*i.e.*, after write events immediately followed by read events).⁶ However, for more complex workflows and actors, read/write transitions alone will not determine state-reset events, and more complex event patterns will be required to accurately describe data dependencies. Figure 3 gives four cases in which read/write transitions do not imply actor transaction boundaries, thus requiring more complex uses of state-reset events.

Figure 3(a) shows an actor A_1 that computes a sequence of running temperature averages (ta_n) from a series of input temperature readings (t_m), along with a corresponding event log for an example run. Each average reading is dependent on all temperature readings received since the most recent state-reset of the actor (*e.g.*, at midnight each night). In the example event log, token ta_{24} depends on tokens t_1 – t_{24} , while ta_{25} is dependent only on t_{25} . Note that assuming that an implicit state-reset follows each write event would be incorrect, because this would imply that each temperature average depended only on the latest temperature reading received, rather than all temperature readings received so far during a particular round of firings.

Figure 3(b) illustrates the necessity of recording state-reset events for a filtering actor. In this example, a series of protein structures are input to actor A_2 , and only those structures meeting a minimum resolution requirement are output (though carried by new tokens). All other input protein structures are discarded by the actor. Thus, in the example event log, of the first six structures received by A_2 , only three are output. Because the state-reset events are recorded,

⁶ Note that state-reset events are still necessary in this example to mark the beginning and end of the actor firing/round.

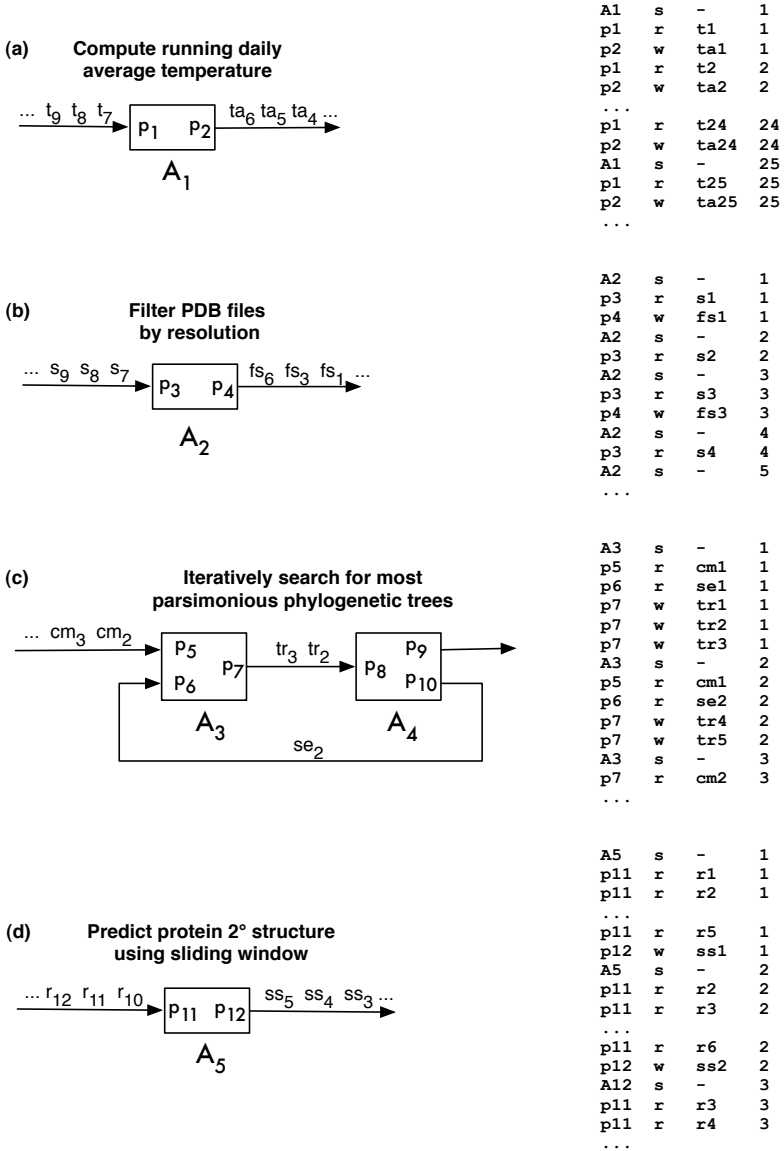


Fig. 3. Four distinct types of actors requiring complex state-reset event behavior

however, it is clear, *e.g.*, that output token fs_3 depends only on input token s_3 , and not on s_2 , even though no write event separates the read events for s_2 and s_3 .

Figure 3(c) illustrates the more general case where an actor reuses only some of the data received during a previous firing. In this example, the tree inference actor A_3 requires a random number seed to initiate a search for maximally

parsimonious phylogenetic trees. Since any particular firing of the actor is not guaranteed to find all of the most parsimonious trees, the actor must be fired iteratively for a particular matrix of phylogenetically informative characters, using a distinct seed on each iteration. Actor A_4 collects the trees inferred by A_3 and provides the seeds needed by A_3 until a sufficient number of trees have been inferred. The RWS model allows each tree inferred in this way to be associated not only with the character matrix from which it was derived, but also with the particular random number seed used by A_3 to discover the tree. The sample event log illustrates how this works. Actor A_3 raises an ‘s’ event prior to receiving each seed, and on receiving that seed declares that it re-reads the character matrix used previously along with the new seed. Thus, it is clear that while trees tr_1 – tr_5 all depend on character matrix cm_1 , only tr_4 and tr_5 were derived using seed se_2 .

Finally, Figure 3(d) illustrates the requirements for recording the provenance of an actor operating on a sliding window of data. Actor A_5 predicts the secondary structure of a protein, residue by residue, based on the types of residues (*i.e.*, amino acids) within a contiguous segment of the protein chain. In this case the RWS model allows the actor to raise an ‘s’ event after writing each output token. The actor then re-reads all tokens except the first token in the current window, along with the next token available on the input, before computing its next output.⁷

3.4 Dependency Graphs

Using the RWS model, we are able to infer from the event log the *token dependency graph*. That is, for each token t , we can know which parent tokens $\{t_1, \dots, t_k\}$ directly contributed to the production of t (as the result of an actor firing). As an example, in the upper left of Fig. 4, $\{t_1, \dots, t_7\}$ are parent tokens of t_{19} . Conversely, t_{22} is the parent of t_{24}, t_{25}, t_{26} . The following Datalog program illustrates how the token dependency graph can be computed from the event log. The *event* relation corresponds to the event log and the *actor* relation contains a mapping from ports to their corresponding actors.

$$\begin{aligned} \text{depends-on}(T_1, T_2) &:- \text{event}(P_1, w, T_1, C_1), \text{event}(P_2, r, T_2, C_2), \\ &\quad \text{actor}(P_1, A), \text{actor}(P_2, A), \text{reset}(A, C_b, C_e), \\ &\quad C_b \leq C_2 \leq C_1 < C_e. \\ \text{reset}(A, C_b, C_e) &:- \text{event}(A, s, -, C_b), \text{event}(A, s, -, C_e), C_b < C_e, \\ &\quad \neg \text{reset-between}(A, C_b, C_e). \\ \text{reset-between}(A, C_b, C_e) &:- \text{event}(-, -, -, C_b), \text{event}(-, -, -, C_e), \\ &\quad \text{event}(A, s, X, C), C_b < C < C_e. \end{aligned}$$

We say that T_1 *depends on* T_2 whenever $\text{depends-on}(T_1, T_2)$ is true.

In addition to the token dependency graph, we are also able to infer the *object dependency graph* using the RWS model. Object dependencies describe user data lineage, and are crucial for our “user-oriented” queries. For example, the middle

⁷ The RWS model could be optimized for cases where actors forget only a small fraction of previously read tokens during each firing by introducing an explicit ‘forget’ event.

column of Fig. 4 shows the object dependencies for the workflow run of Fig. 1. Note that the object dependency graph differs slightly from the token dependency graph. Object dependency graphs can be computed from corresponding token dependency graphs and token-object mappings.

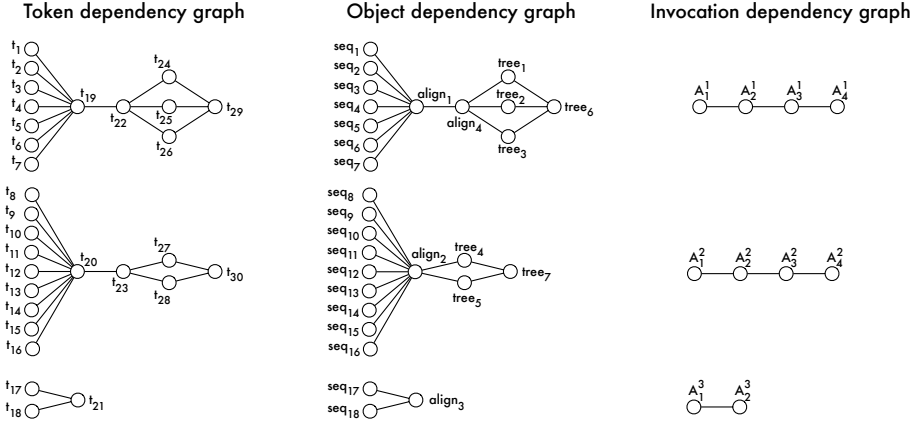


Fig. 4. Token, object, and actor invocation graphs for our example phylogenetics workflow. Dependencies are shown from left to right. Note that all but one of the token-object mappings can be inferred from the graph structures; tokens t_{20} and t_{23} both map to the object $align_2$.

Finally, *actor-invocation dependency graphs* can also be inferred directly from event logs in the RWS model. In particular, this graph can be built from state-reset events in the event log such that an actor invocation A_2^j depends on another actor invocation A_1^i whenever A_2^j reads a token that is written by A_1^i . Note that here, “invocation” refers to a firing round. It should be clear that all of the information stored in the event table is required to reconstruct these token, object, and invocation dependency graphs for a workflow trace. In particular, if state-reset events are not taken into account, each token written by an actor will (incorrectly) *appear* to depend on *all* previous tokens read during prior firing rounds of the actor: *e.g.*, in the absence of state-reset events, t_{21} would be connected to tokens t_1 to t_{18} in the token dependency graph of Fig. 4.

4 Querying Workflow Traces

A wide range of scientifically relevant questions can be answered using the provenance model described above. To make access to event logs more convenient, we introduce the following primitive operations, which can be implemented, *e.g.*, as relational selections over the event log. The *writer(t)* and *reader(t)* operations return the ports that a token t was written to and read from, respectively (a token is written to a port exactly once, but can be read multiple times). The

$token\text{-}parents(t)$ and $token\text{-}children(t)$ operations return the set of direct token dependencies for a token t , while $token\text{-}ancestors(t)$ and $token\text{-}descendants(t)$ are their transitive closures. The $siblings(t)$ operation returns the tokens with the same direct dependencies as t ; *e.g.*, because actor A3 can infer multiple trees from an alignment, given one of these trees, $siblings$ returns the other trees computed from the same alignment. The $origin(o)$ and $death(o)$ operations return the first and last tokens in the trace that refer to the object o ; *e.g.*, the origin and death operations can be used to determine that the alignment object $align_2$ originated with token t_{20} (written by actor A1) and terminated with token t_{23} (written by actor A2).

The following examples illustrate how the provenance operations can be combined to answer concrete questions of interest to a scientist using the workflow in Figure 1. For each high-level question below, we define a corresponding parameterized query using set-comprehension syntax⁸, along with the actual results for the event log given in Figure 2. Below, we use \mathcal{W} to denote the workflow graph (in Figure 1) and \mathcal{T} for the corresponding trace.

- **What DNA sequences were input to the workflow?** This is one of the first questions a scientist might ask about the workflow run. Given an object type $\$c$, the parameterized query

$$q_1(\$c) := \{o \mid t \in tokens(\mathcal{T}) \wedge writer(t) \in in(\mathcal{W}) \wedge object(t) = o \wedge \$c \in types(o)\},$$

returns the set of objects of type $\$c$ that were input to the workflow run. For our example trace, $q_1(\text{SEQUENCE})$ returns the objects seq_1 to seq_{18} . The expression $t \in tokens(\mathcal{T})$ selects a token from the trace, the expression $writer(t) \in in(\mathcal{W})$ checks that the token was written by an input port of the workflow \mathcal{W} , the expression $object(t) = o$ obtains the object associated with t , and the expression $\$c \in types(o)$ verifies that o has $\$c$ as a type.

- **What phylogenetic trees were output by the workflow?** This is another basic question that a scientist might initially ask after a run. Given the query

$$q_2(\$c) := \{o \mid t \in tokens(\mathcal{T}) \wedge reader(t) \in out(\mathcal{W}) \wedge object(t) = o \wedge \$c \in types(o)\},$$

the expression $q_2(\text{TREE})$ returns the objects $tree_6$ and $tree_7$.

- **What phylogenetic trees (intermediate or final) were created by the workflow?** This question requests both intermediate as well as final data products of a run. Given the query

$$q_3(\$c) := \{o \mid t \in tokens(\mathcal{T}) \wedge writer(t) \notin in(\mathcal{W}) \wedge object(t) = o \wedge \$c \in types(o)\},$$

the expression $q_3(\text{TREE})$ returns all tree objects of Figure 4. Note that the expression $writer(t) \notin in(\mathcal{W})$ ensures that the returned trees were not given as input to the workflow.

⁸ Queries could also be defined in Datalog or in query languages for graphs or semi-structured data.

- **What actor created this phylogenetic tree?** The following query returns the actors that first wrote the given object $\$o$:

$$q_4(\$o) := \{a \mid t \in \text{origin}(\$o) \wedge \text{actor}(\text{writer}(t)) = a\}.$$

The query returns A_3 for tree_1 to tree_5 , and A_4 for tree_6 and tree_7 . This question is of particular interest for workflows that employ multiple approaches for inferring phylogenetic trees.

- **Which phylogenetic trees were directly used to compute this consensus tree?** This question (*i.e.*, what is this tree the “consensus” of?) asks for the intermediate data products supplied to the actor producing a particular workflow output. Given the query

$$q_5(\$c, \$o) := \{o' \mid t \in \text{origin}(\$o) \wedge t' \in \text{token-parents}(t) \wedge \text{object}(t) = o' \wedge \\ \$c \in \text{types}(o')\},$$

the expression $q_5(\text{TREE}, \text{tree}_6)$ returns tree_1 to tree_3 ; and $q_5(\text{TREE}, \text{tree}_7)$ returns tree_4 to tree_5 .

- **What sequences input to the workflow does this consensus tree depend on?** This question illustrates how a workflow output can be related to the particular workflow inputs from which it was derived. Given the query

$$q_6(\$c, \$o) := \{o' \mid t \in \text{origin}(\$o) \wedge t' \in \text{token-ancestors}(t) \wedge \text{writer}(t') \in \text{in}(\mathcal{W}) \wedge \\ \text{object}(t') = o' \wedge \$c \in \text{types}(o')\},$$

the expression $q_6(\text{SEQUENCE}, \text{tree}_6)$ returns seq_1 to seq_7 , and the expression $q_6(\text{SEQUENCE}, \text{tree}_7)$ returns seq_8 to seq_{16} .

- **Which input sequences were not used to derive any output consensus trees?** Here we are interested in whether there are any workflow inputs without corresponding workflow outputs. Such inputs may be considered the workflow equivalent of “phantom lineages” [23]. Given an input type $\$c_{in}$ and output type $\$c_{out}$, the query

$$q_7(\$c_{in}, \$c_{out}) := \{o \mid t \in \text{tokens}(\mathcal{T}) \wedge \text{writer}(t) \in \text{in}(\mathcal{W}) \wedge \text{object}(t) = o \wedge \\ \$c_{in} \in \text{types}(o) \wedge \{t' \mid t' \in \text{token-descendants}(t) \wedge \\ \text{reader}(t') \in \text{out}(\mathcal{W}) \wedge c_{out} \in \text{types}(\text{object}(t'))\} = \emptyset\},$$

returns the objects input to the workflow that do not produce any workflow outputs; *e.g.*, the expression $q_7(\text{SEQUENCE}, \text{TREE})$ returns the sequences seq_{17} and seq_{18} . The query first finds workflow input tokens t that refer to objects of type $\$c_{in}$, and then checks (via a subquery) to make sure that t has no output tokens with objects of the type $\$c_{out}$.

- **What was the sequence alignment used in the process of inferring this tree?** This question requests the key intermediate data object used in producing a workflow result. A researcher may wish to examine the alignment to assess the reliability of the results, or reuse the alignment in another workflow.

Given the query

$$q_8(\$c, \$o) := \{o' \mid t \in \text{origin}(\$o) \wedge t' \in \text{token-ancestors}(t) \wedge \text{object}(t') = o' \wedge \\ \$c \in \text{types}(o') \wedge \{t'' \mid t'' \in \text{token-descendants}(t') \wedge \\ \$c \in \text{types}(\text{object}(t''))\} = \emptyset\},$$

the expression $q_8(\text{ALIGNMENT}, \text{tree}_6)$ returns the sequence alignment align_4 , and $q_8(\text{ALIGNMENT}, \text{tree}_7)$ returns the sequence alignment align_2 . The subquery above ensures that the object o' is the alignment directly used to infer the tree.

• **What actors were involved in creating this tree?** This question may be used, *e.g.*, when writing the methods section of a publication to cite the employed methods and implementations. Given the query

$$q_9(\$o) := \{a \mid t \in \text{origin}(\$o) \wedge \text{actor}(\text{writer}(t)) = a\} \cup \\ \{a \mid t \in \text{origin}(\$o) \wedge t' \in \text{token-ancestors}(t) \wedge \text{actor}(\text{writer}(t')) = a\},$$

the expression $q_9(\text{tree}_6)$ returns actors A1 to A4.

• **Which actors did not produce any output for input derived from this input sequence?** This question provides an explanation for the phantom lineages revealed by q_7 above:the query

$$q_{10}(\$o) := \{a \mid t \in \text{origin}(o) \wedge t' \in \text{token-descendants}(t) \wedge \text{token-children}(t') = \emptyset \\ \wedge \text{actor}(\text{reader}(t')) = a\},$$

The expressions $q_{10}(\text{seq}_{17})$ and $q_{10}(\text{seq}_{17})$ both return actor A2, indicating that this actor did not forward a refined sequence alignment of these two sequences to actor A3. This result is reasonable since no informative phylogenetic trees may be inferred from only two taxa.

5 Conclusion

Tracking provenance is an important aspect of scientific workflow systems. In this paper, we have focused primarily on the problem of tracking data lineage within scientific workflow runs, for the purpose of providing an accurate provenance record for answering “scientific” (*i.e.*, user-oriented) provenance queries.

The problem of data lineage has been widely studied in the database community [5,8,2,23]. However, the primary focus has been on transformations of data items expressed as database queries. As noted in [11], current provenance approaches for workflow systems (*e.g.*, [24,25,18]) record various kinds of meta-data related to provenance. Despite these developments, however, little support exists in current systems to allow end-users to query provenance information in *scientifically meaningful* ways, in particular when advanced workflow execution models go beyond simple DAGs (as in process networks).

We have shown that a simple provenance model, based on read, write, and state-reset events, is expressive enough to capture many relevant science-oriented provenance use cases. These use cases become queries against suitable views on

top of the event log. Our approach also marks the beginnings of a use-case and computation-model driven approach to provenance schema design. Using our framework, it is now meaningful to ask whether a provenance schema can handle specific use cases, since the latter become queries over the former.

As future work we intend to extend our approach to support a wider array of operations, *e.g.*, so-called “smart re-runs” (a workflow system requirement in [16])⁹ and crash recovery, and to extend our current Prolog-based prototype to provide direct support (including query user interfaces) for our provenance model within KEPLER. We are also developing methods to optimize our approach to reduce the size of event logs for actors whose behaviors are similar to sliding window operators (*e.g.*, by introducing a “forget” event), and to support subworkflows within KEPLER (*i.e.*, composite actors), *e.g.*, by inferring in a bottom-up fashion the appropriate state-reset events for the composite actor via the state-reset events of subsumed actors and the corresponding workflow graph.

References

1. D. Berry, P. Buneman, M. Wilde, and Y. Ioannidis, editors. *e-Science Workshop on Data Provenance and Annotation*, National e-Science Centre, Edinburgh, December 2003.
2. D. Bhagwat, L. Chiticariu, W. C. Tan, and G. Vijayvargiya. An annotation management system for relational databases. In *Proc. of VLDB*, 2004.
3. R. Bose and J. Frew. Lineage retrieval for scientific data processing: A survey. *ACM Computing Surveys*, 37(1):1–28, 2005.
4. P. Buneman and I. Foster, editors. *Workshop on Data Derivation and Provenance*, Chicago, October 2002.
5. P. Buneman, S. Khanna, and W. C. Tan. Why and where: A characterization of data provenance. In *Proc. of ICDT*, volume 1973 of *LNCS*, 2001.
6. S. P. Callahan, J. Freire, E. Santos, C. E. Scheidegger, C. T. Silva, and H. T. Vo. Managing the evolution of dataflows with vistrails. In *IEEE Workshop on Workflow and Data-Flow for Scientific Applications (SciFlow)*, 2006.
7. J. Castresana. Selection of conserved blocks from multiple alignments for their use in phylogenetic analysis. *Mol. Biol. Evol.*, 17:540–552, 2000.
8. Y. Cui and J. Widom. Lineage tracing for general data warehouse transformations. In *VLDB*, 2001.
9. Y. Cui, J. Widom, and J. Wiener. Tracing the lineage of view data in a warehousing environment. *ACM TODS*, 25(2), 2000.
10. E. Deelman, J. Blythe, Y. Gil, C. Kesselman, G. Mehta, S. Patil, M.-H. Su, K. Vahi, and M. Livny. Pegasus: Mapping scientific workflows onto the grid. In *Proc. of the European Across Grids Conference*, 2004.
11. C. Goble. Position statement: Musings on provenance, workflow and (semantic web) annotations for bioinformatics. In Buneman and Foster [4].
12. G. Kahn and D. B. MacQueen. Coroutines and networks of parallel processes. In *Proc. of the IFIP Congress*, 1977.

⁹ Specialized provenance systems for smart re-runs exist already [6].

13. E. A. Lee and D. Messerschmitt. Static scheduling of synchronous data flow programs for digital signal processing. *IEEE Transactions on Computers*, C-36, 1987.
14. E. A. Lee and S. Neuendorffer. Actor-oriented models for codesign: Balancing reuse and performance. In *Formal Methods and Models for System Design*. Kluwer, 2004.
15. E. A. Lee and T. M. Parks. Dataflow process networks. *Proc. of the IEEE*, 83(5), 1995.
16. B. Ludäscher, I. Altintas, C. Berkley, D. Higgins, E. Jaeger, M. Jones, E. A. Lee, J. Tao, and Y. Zhao. Scientific workflow management and the Kepler system. *Concurrency and Computation: Practice & Experience*, 2005.
17. L. Moreau, O. Rana, and D. Walker. Provenance aware service-oriented architecture (pasoa). pasoa.org, 2006.
18. T. M. Oinn, M. Addis, J. Ferris, D. Marvin, M. Senger, R. M. Greenwood, T. Carver, K. Glover, M. R. Pocock, A. Wipat, and P. Li. Taverna: A tool for the composition and enactment of bioinformatics workflows. *Bioinformatics*, 20(17), 2004.
19. PHYLIP Phylogeny Inference Package. <http://evolution.gs.washington.edu/phylip.html>.
20. Y. Simmhan, B. Plale, and D. Gannon. A survey of data provenance in e-science. *SIGMOD Record*, 34(3):31–36, September 2005.
21. D. Thain, T. Tannenbaum, and M. Livny. Distributed computing in practice: The Condor experience. *Concurrency – Practice and Experience*, 17(2-4), 2005.
22. J. D. Thompson, D. G. Higgins, and T. J. Gibson. CLUSTAL W: Improving the sensitivity of progressive multiple sequence alignment through sequence weighting, position specific gap penalties and weight matrix choice. *Nucleic Acids Res.*, 22:4673–80, 1994.
23. J. Widom. Trio: A system for integrated management of data, accuracy, and lineage. In *Conference on Innovative Data Systems Research (CIDR)*, 2005.
24. S. Wong, S. Miles, W. Fang, P. Groth, and L. Moreau. Provenance-based validation of e-science experiments. In *ISWC*, 2005.
25. J. Zhao, C. Goble, R. Stephens, and S. Bechhofer. Linking and browsing provenance logs for e-science. In *ICSNW*, 2004.

Applying the Virtual Data Provenance Model

Yong Zhao¹, Michael Wilde², and Ian Foster²

¹ University of Chicago

yongzh@cs.uchicago.edu

² University of Chicago and Argonne National Laboratory

Abstract. In many domains of science, engineering, and commerce, data analysis systems are employed to derive new data (and ultimately, one hopes, knowledge) from datasets describing experimental results or simulated phenomena. To support such analyses, we have developed a “virtual data system” that allows users first to define, then to invoke, and finally explore the provenance of procedures (and workflows comprising multiple procedure calls) that perform such data derivations. The underlying execution model is “functional” in the sense that procedures read (but do not modify) their input and produce output via deterministic computations. This property makes it straightforward for the virtual data system to record not only the recipe for producing any given data object but also sufficient information about the environment in which the recipe has been executed, all with sufficient fidelity that the steps used to create a data object can be re-executed to reproduce the data object at a later time or a different location. The virtual data system maintains this information in an integrated schema alongside semantic annotations, and thus enables a powerful query capability in which the rich semantic information implied by knowledge of the structure of data derivation procedures can be exploited to provide an information environment that fuses recipe, history, and application-specific semantics. We provide here an overview of this integration, the queries and transformations that it enables, and examples of how these capabilities can serve scientific processes.

1 Introduction

We present a general model for representing and querying provenance information within the context of a Virtual Data System (VDS) that captures, and enables discovery of, the relationships among data, procedures and computations. We focus, in particular, on the VDS query model, and examine how knowledge of the provenance of virtual data objects and their relationships can be used to enhance program development, data analysis, and other tasks.

In what we call the *virtual data model*, we associate with each data object the functional procedure that was used, or can be used, to produce or reproduce it. Such associations are defined with sufficient fidelity that the steps used to create a data object can be re-executed to reproduce the data object (within obvious limitations) at a later time or a different location. We refer to the information that we record to achieve this reproducibility the *provenance* of a data object. (Throughout this article,

we use the term “procedure” to denote executable application programs, but the paradigm applies equally well to a service-oriented model in which “procedures” correspond to invocations of remote operations.)

We view provenance in this context as comprising two parts: all the aspects of the procedure or workflow used to create a data object (*prospective* provenance, or “recipe”) as well as information about the runtime environment in which a procedure was executed and the resources used in its invocation (*retrospective* provenance).

While only the prospective information is needed to produce or reproduce a data object, we argue that the *complete* provenance record—prospective and retrospective—provides a more complete understanding of the data. For instance, retrospective provenance can help investigate a data derivation process, as it keeps information regarding the environment in which the process was performed. This level of understanding is of great value in scientific data preparation and analysis, allowing the user to (for example) reason about the validity of data and conclusions drawn from it; determine and assess the methods that were used to process the data; and transform or compose existing methods to handle new problems.

The Virtual Data System that we have developed to implement this model [ZW+05] maintains a precise record of procedures, inputs (both data and parameter settings) to procedures, the environment in which procedures were invoked, and relevant data about how a procedure behaved (e.g., duration). Armed with this information, we can track, for any data object created within the system, a derivation history that extends back to raw input data, and thus obtain accurate and complete information about how analysis conclusions (and all intermediate results) were derived. We can understand data dependencies, and reason about the consequences for an analytical finding of changing some processing step, parameter, or input dataset. We can audit how results were derived, and create new recipes for conducting new investigations that build on previous findings and approaches.

An important component of VDS is the Virtual Data Language (VDL) [FV+02], a functional scripting language that we use to describe relations among data, procedures, and computations that invoke procedures. A data analysis workflow expressed in VDL makes the relationships among these different elements explicit.

VDS also incorporates sophisticated mechanisms for executing both individual procedures and more complex workflows in distributed environments. These mechanisms include tools for integrating data in diverse physical representations [ZD+05], workflow transformation tools and planners, such as the Pegasus system [DS+05, SKD06], the DAGman workflow execution system [FT+02], and Globus mechanisms for secure and reliable remote execution and distributed data management [F05]. This aspect of the system is less relevant to our goals here (except in that the transformations performed, and specific execution sites chosen, may be an important part of the provenance record), and so we do not discuss it further in this article.

VDS is distinguished from other approaches to provenance recording by its focus on a particular computational model, namely the functional model defined by VDL. While this focus restricts the set of computations that can be represented, we do not find this restriction to be onerous in practice, and the benefits in terms of the depth of provenance information that can be captured efficiently and the variety of queries that can be posed against that data are significant.

We focus in this article on illustrating the virtual data approach to integrating prospective and retrospective provenance with semantic annotations; describing the powerful queries that can be performed on such an integrated base; and introducing the implementation techniques that can provide these benefits in a large-scale scientific computing environment. The basic mechanisms for these techniques have been implemented in our Virtual Data System for some time [FV+02]; this paper describes schema extensions whose implementation is in progress.

2 Virtual Data Schema for Provenance Recording

We model as a logical virtual data schema the various relationships that exist among *datasets*, *procedures*, *calls* to procedures (which operate on datasets), and the zero or more physical *invocations* of a specific call. These relations are described by the entity-relationship (ER) diagram of Figure 1. In this diagram, primary keys are underlined; foreign keys are implied by graph edges.

A computational *procedure* represents an application or a service that can be executed or invoked. A procedure definition describes the procedure’s signature: its name and formal arguments. A procedure may be defined in a specific namespace and may have different versions. Procedures are therefore identified by the 3-tuple (*namespace*, *name*, *version*). A formal argument (*FormalArg*) has a name, a type, and a direction attribute that indicates whether it is an input or output argument.

A *call* specifies a procedure call, supplying actual arguments (*ActualArgs*) that bind values and datasets to formal arguments. Like procedures, calls are identified by (*namespace*, *name*, *version*) tuples, but for calls these unique tuples can be generated

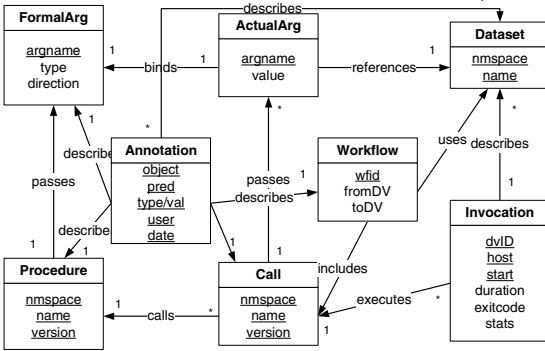


Fig. 1. Schema for provenance and annotation

automatically. A call statement is considered prospective as it only declares a way to invoke a procedure, and specific data products can be generated by making the described procedure call. But by itself, a call is not executed until it is included in a *workflow* and scheduled to run. We use the term *derivation* (DV) interchangeably for a call.

We also model *workflows*, i.e., sets of calls that operate on the same datasets. A workflow is represented by one or more entries in the Workflow table, each entry for an edge of the workflow graph, which contains the source call (*fromDV*) that produces a certain data product, and the target call (*toDV*) that consumes that data product. A workflow itself, like a call, is prospective, and can be enacted multiple times. Each enactment of both a call and a workflow is recorded by an invocation record.

This integration of workflows enables queries to consider the provenance history of data objects, and the relationships among procedures based on the patterns in which they are actually used in defined workflows.

Metadata associates annotations with datasets (via their names), procedures and arguments, calls, and workflows. Annotations take the form of a named predicate and a typed value (a string, integer, float, boolean, or date). This simple type model for annotation values is readily extended to use, for example, the flexible data typing model defined by XML Schema. Such metadata annotations are similar to RDF triples: The *subject* is one of the 5 entities in the virtual data provenance model that can be annotated (datasets, procedures, etc.); the *predicate* is the “name” of the annotation (i.e., the assertion being made) and the value is the *object*. We plan to extend annotations with user and date information, so that we can maintain the provenance of metadata itself.

3 General Model for Provenance and Annotation Query

Having defined our virtual data model in relational terms, we can use standard SQL to query entities in the data model. For example, we can ask questions such as “select procedure calls whose argument *modelType* has value *nonlinear*,” “select invocations that ran at location *Argonne*,” and the join query “select procedure calls that ran at location *Argonne* and whose argument *modelType* has value *nonlinear*.”

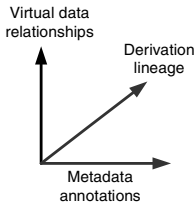


Fig. 2. Query dimensions

We find it useful to think of the virtual data query model as having three major dimensions (see Figure 2): (1) prospective and retrospective provenance data, as provided by records of procedure definition, procedure arguments, and runtime invocation recording; (2) metadata annotations that enrich this application-independent schema with application-specific information; and (3) lineage information

obtained by interrogating the patterns of procedure calls, argument values, and metadata inherent in the workflow graphs that describe the indirect nature of the production of a given data object. We describe below the general nature of these three dimensions, and then discuss how queries can be defined that join across these dimensions. We provide detailed query examples drawn from scientific applications in Section 4 and Section 5.

Virtual Data Relationship Queries. The core queries in our model are based on the fundamental entities of the virtual data schema: the prospective declarations of procedure definitions and calls and the retrospective records of actual procedure invocations. These queries focus on the primary tables of the virtual data schema. The following examples illustrate the range of queries that are supported. The first two forms of queries deal with prospective information, while the third deals with retrospective information.

Fundamental Queries of Entity Attributes: Find procedures and calls by namespace, name, and version; find all the calls that invoke a given procedure.

Query by Parameters: Find procedures that pass a specified parameter; find procedure calls that pass a parameter of a specified type and/or value in a specified direction (i.e., input or output); find invocations that executed with a specified parameter value and direction; find procedure calls that process a specified file as input or output; find all files consumed or returned by a specified procedure call.

Query of Invocation Records: find invocation records by procedure or procedure-call namespace, name, version; find procedures or procedure calls executed at a specified site; find procedures or calls executed at a specified host; find invocations run on machines with a specified OS type; find jobs with a specified exit status; find jobs with run time $>$ a specified r ; find jobs within a set of jobs that ran longer than twice that the set's average time; find invocations that produce files of a specified type and size; find the invocation records that produce or consume a specified dataset.

Annotation Queries. The annotation capabilities provided by the virtual data model on procedures, arguments, calls, datasets, and workflows form the basis for the second query dimension. While various applications may use these annotations to maintain application-specific provenance, we consider this a separate dimension from the provenance information that is intrinsic to the virtual data model.

Annotation queries can, for example, select all annotations for any annotatable virtual data object or set of objects, or select from an annotation result set based on any of subject, predicate, object, object type, user, or annotation date.

Annotations can also be used to select virtual data objects: for example, find all objects (of any type) annotated with predicate p of type t and value v ; objects of a specific type annotated with predicate p of type t and value v ; or objects (one type or any type) annotated by same set of attribute predicates.

Lineage Graph Queries. A powerful source of information in a virtual data system is the lineage relationships [WS97] that we can derive for all data products. For example, knowing that the inputs to a procedure A_k were processed by A_i can often tell a scientist important characteristics about the results that A_k will derive. Knowing further whether A_j processed the output of A_i somewhere between those two steps may determine whether further analysis of that chain of data is required.

A simple class of lineage graph queries refers to information that has been propagated along derivation relationships. For example, “find datasets derived from dataset d ” or “find ancestor datasets to dataset d that have type t .”

More complex queries may refer to patterns within the derivation graph. Much work has been done in this field: for example, Giugno and Shasha [GS02] describe a model for such patterns in a system called GraphGrep. We propose to adapt the GraphGrep model here to the specific problem of matching workflow graphs. We sketch here how we expect to apply this model to enable pattern-based searches of derivation graphs.

The basic approach is to introduce special objects that can match specific patterns of procedures, calls, and invocations, enabling the composition of “workpattern” objects that can perform powerful searches and queries on the workflows in our database. The semantics of such matches work as follows. Procedure patterns, call patterns, and invocation patterns, chained into a DAG within a workpattern object, can match either fixed or varying numbers of nodes of their corresponding object types in any workflow defined in the database. The nodes of a workpattern graph can

match procedure definitions or calls that meet criteria such as argument name, argument values, argument types, and/or annotations.

Performing a query on a workpattern can select a set of workflows, where in each selected workflow one or more subgraphs are matched. The target search space of a workpattern query can be either the entire database, or a specific workflow or set of workflows selected through a prior search. Using the query model defined above, we can perform queries such as: find datasets that were derived within N levels of procedure p ; find datasets that are the result of workpattern wp ; and find the procedure calls in workflow w whose inputs have been processed by any workflow matching workpattern wp .

Provenance Queries in Multiple Dimensions. The capabilities of the queries defined above are amplified by the ability to join them flexibly across multiple dimensions of the virtual data schema. For example, we may ask for procedures with a specified signature that have been called with specific argument values (or ranges) and that match an annotation query; the metadata values for a specified set of predicates from a list returned by another query; or the minimum, maximum, and average run times of a set of procedure calls matching workpattern wp and annotation query q .

For example, a set of procedures selected by a workpattern query can be used to select metadata values that are then used as a search key to select a set of procedure calls. This level of nesting can be used to successively filter (or expand) a result set, and such query chaining can take place to effectively arbitrary depths (limited only by the capabilities of the underlying database system).

Modification and Composition Queries. Maintaining dataset, procedure, workflow, annotation, and provenance information in an integrated schema facilitates not only powerful queries, but also the ability to couple queries with database update procedures to define new procedures, annotations, and work requests. We illustrate such possibilities below.

Change Arguments: For every procedure call $p1$ to procedures in namespace n with annotation m , create a new procedure $p2$ with argument a replaced by an expression e .

Change Procedures: In every workflow w matching workpattern wp , create a new workflow with the same name but a new version number in which procedure $p1$ is changed into procedure $p2$ (which must have the same signature).

Edit Subgraphs of a Workflow: In every workflow w matching workpattern wp , create a new workflow with the same name but a new version number in which the matching workpattern subgraph is changed to a specified new workflow subgraph. (The supplied replacement workflow subgraph must have the same signature.)

Replicate a Workflow: Given a workflow w to replicate, for each procedure $p2$ returned by query Q , create a new workflow $w2(p2)$ by replacing occurrences of p in w with $p2$. Each $p2$ returned by Q must have the same signature as p .

Edit Metadata: In every workflow w matching workpattern wp , edit annotations on datasets output by the subgraph matched by wp , changing the value of metadata predicate p to a new value nv .

4 Query Examples Drawn from fMRI Science Use Cases

The capabilities that we have described are only interesting if they provide utility to users addressing real data analysis problems. In this section we show such use cases, drawing examples from the field of functional MRI research [HSM02], in which MRI brain images of some subjects are firstly spatially aligned, and then averaged to produce a single image. The procedures employ the AIR (automated image registration) suite [WG+98a,WG+98b] to create an averaged brain from a collection of high resolution anatomical data. The images are annotated with metadata tags such as *studyModality*, *center*, and *state*. We use the categorization of queries introduced earlier in Section 3.

Virtual Data Relationship Queries

- Find all the procedures in namespace */pub/bin/std* that have inputs of type *SubjectImage* and outputs of type *ThumbNailImage*.
- Find all *alignlinear* calls (including all arguments), in XML format, with argument *model=rigid*, and that generated more than 10,000 page faults, on *ia64* processors.
- Find all calls to procedure *alignlinear*, and their runtimes, with argument *model=rigid* that ran in less than 30 minutes on non-*ia64* processors.
- Find the average runtime of all *alignlinear* calls with argument *model=rigid* that ran in less than 30 minutes.
- Find all procedure calls within workflow */prod/2005/0305/prep* whose inputs were linearly aligned with *model=affine*

Annotation Queries

- Find all the datasets that have metadata annotation *studyModality* with values *speech*, *visual* or *audio*. Show all the annotation tags of this set of datasets.
- Show the values of all annotation predicates *developerName* of procedures that accept or produce an argument of type *Study* with predicate *studyModality=audio*.

Lineage Queries

- Given the workpattern:
 $alignlinear(model=affine) \Rightarrow reslice(axis=x, intensify=3) \Rightarrow softmean$
 find all output datasets produced by *softmean* calls that were linear-aligned with *model=affine*. (I.e., “where *softmean* was preceded in the workflow, directly or indirectly, by an *alignlinear* call with argument *model=affine*”)
- Find all output datasets of *softmean* that were resliced with *intensify=3*. (Here we want a *softmean* that is directly preceded by the requested pattern.)

Combined Queries

- Find procedures that take an *ImageAtlas* dataset and a *Date* as arguments, have been called with dataset *atlas.std.2005.img*, and have annotation *QALevel* with value > 5.6 .
- Find all metadata tags *studyModality* on result datasets that were linearly aligned with parameter *model=affine* and with an input dataset annotated with *center* set to *UChicago*.

- Find the output dataset names (and all their metadata tags) that were linearly aligned with *model=affine* and with input file metadata *center=UChicago*.
- Find all the metadata tags *center* with values in the set (*UIUC, UChicago, UIC*) of output datasets of *softmean*.
- Find all the metadata tags *center* with values in set (*UIUC, UChicago, UIC*) of outputs of *softmean* that were aligned with *model=affine*.
- Find all the metadata tags *studyModality* on results of *softmean* that were linearly aligned with *model= affine*, and whose output datasets have annotation *state = IL*.

As these examples show, our provenance architecture allows for the expression of a wide range of interesting queries. We need to conduct further experimentation with additional applications, larger and more diverse user communities, and larger data collections to verify that we can both pose and answer efficiently the questions that users want to ask in practice.

5 Implementation and Experience

The VDS implementation of virtual data mechanisms allows for declarative specification of data, procedures, computations and their relationships, using VDL. VDL definitions and provenance data are stored in a “virtual data catalog” (VDC), typically implemented as a relational database and accessed via SQL. We employ an adapter layer to allow the use of different relational database implementations, and support the use of XML databases for the VDC. The actual physical schema used in our implementation is slightly more complex than the logical-level model shown in Figure 1, but captures essentially the same information. The graph structure of workflow objects is currently maintained in an external XML document. VDC queries are parsed and translated into SQL or XQuery/XPath statements to apply against the VDC database.

The physical schema uses a separate value table for each of the five metadata value types supported (string, int, float, date, boolean). This approach enables us to utilize native database searches that treat the data type of the object properly and efficiently (e.g., proper comparison and collation for floating point numbers and dates).

VDS translates requests to derive virtual data products into workflows that may execute at multiple distributed locations. Runtime provenance is obtained by executing VDL procedures under a uniform parent-process wrapper that collects information about the execution of the child application, and its derived files, using OS services. This information is then routed back to the workflow enactment engine via embedded steps in the workflow and saved in the virtual data catalog as invocation records.

An example of the use of virtual data provenance recording is seen in the analysis of provenance information captured by the ATLAS high energy physics experiment to generate simulated events using VDS from 6/2004 through 12/2005. In this period, 20 different simulation procedures were defined in a central US-ATLAS VDC located at Brookhaven National Lab. This virtual data catalog captured 1.2M run-time (retrospective) provenance records, of which 574K described procedure invocations

detailed in the same number of prospective provenance records in the database. 447K unique simulation datasets (logical files) were derived from these invocations.

We can probe the provenance in this catalog with queries that physicists can usefully employ to search for and assess these simulation results. Questions like the following (translated to SQL) can be easily answered (with actual results shown):

Q: List all the procedures that have argument name 'cleanLevel':

```
=> brureconx evgenx ... G4simulx g4simx g4simxM pileup testreconx
```

Q: How many jobs running procedures with argument name 'cleanLevel' were run on Linux 2.4.28 kernels?

```
=> g4digitx 39
    g4simx 340
```

Q: List calls of procedure 'g4simx' with argument eta_min=-5.0 and eta_max=5.0 that were run on 2.4.28 kernels, in Dec 2004?

```
=> g4simx.CPE_4922_15
    g4simx.CPE_4922_202
    ... (total 285 calls)
```

Simple aggregations and statistics can also be carried out over the records, for reporting purposes, as in the following examples.

Q: List the total number of jobs run in each month of 2004:

year	month	number_of_jobs
2004	06	1433
2004	07	13331
2004	08	21076
2004	09	20807
2004	10	32364
2004	11	39681
2004	12	14734

Q: List the total run time (in unit of year) of jobs run in each month of 2004:

year	month	run_time(years)
2004	06	0.4
2004	07	20
2004	08	34
2004	09	40
2004	10	15
2004	11	15
2004	12	8.9

Another application of VDS to capture and leverage provenance information is in the QuarkNet nationwide physics education project [BG+05]. In this project, data from cosmic ray detectors located in about 200 high schools in the US uploaded raw data into a data analysis portal driven by VDS. The raw data was annotated and then processed with a set of analysis tools to plot cosmic ray activity under a variety of experimental conditions and derive and document scientific conclusions, modeling closely the processes used in experimental physics collaborations. In this application trial 108 different procedures were used to process 6,330 files (total raw and derived) and to annotate them with 134,834 metadata tuples. A sample query of the

annotations on a data file (a flux study result derived from data gathered by detector 180 channel 1 on 07/30/2004) yields:

```
project:      cosmic           city:         Batavia
group:       fermigroup      state:        IL
study:       flux            creationdate: 2005-01-13 17:44:20.512
detectorid:  180             rawdate:      2004-07-30 19:42:57.0
```

A query to select datasets based on annotations, such as “find all the *blessed* data from *Fermilab*” is expressed in SQL as:

```
select name from anno_lfn f, anno_bool b where f.mkey='blessed' and
b.value=true and f.id=b.id intersect select name from anno_lfn f2,
anno_text t where f2.mkey='school' and t.value='Fermilab'
and f2.id=t.id
```

which returns:

```
180.2004.0730.35
...
999.2005.0604.0
(total 5 rows)
```

6 Related Work

Work on provenance in database systems has focused on determining the source data (tuples) used to produce an item. Cui and Widom [CW00, CWW00] record the relational queries used to construct materialized views in a data warehouse, and then exploit this information to find the source data that contributed to the given data item. Buneman et al. [BKT01] distinguish between *why-provenance* and *where-provenance*. The former explains why a piece of data is in the database, i.e., what data sets (tuples) contributed to a data item, while the latter keeps track of the location of a data item in its source. The mutability of database tables and records poses significant challenges. In contrast, we address provenance issues within the context of data analyses performed using programs that are assumed not to modify their input datasets. In this context, we can go beyond *why* and *where* to address issues of *how* a data product was (or can be) derived, *what* are the procedure definitions and annotations, and to *which* workflow the procedure belongs.

Various systems support provenance tracking in the scientific community. SAM [MC+03] has a “laboratory notebook” model of provenance tracking in which metadata can be added to data items stored in a repository. However, SAM does not define the format or schema of the metadata or an underlying computational model. The same is true of other notebook schemes [BK+06]. In myGrid, documentation about workflow execution is recorded and stored in a user’s personal repository, along with other metadata [ZG+03,ZGR06], to support personalized provenance tracking of bioinformatics services and workflows.

Szomszor and Moreau [SM03] propose a service-based architecture for recording provenance in a Grid environment. They rely on a workflow enactment engine to submit service invocation information to a provenance service. Moreau et al. [GM+05] describe an implementation-independent architecture for provenance systems. They describe a logical architecture in which so-called *p-assertions* can be

submitted and retrieved from a *p-store* by various actors, and a process architecture for system security and distribution. This system has been applied to physics [BM06], engineering [KS06], and transplant management [AV+06], among others.

Our approach is distinguished from that of Moreau et al. by its integration of provenance with a particular computational model, namely that captured by our functional virtual data language. This model makes it feasible for us to capture high-fidelity retrospective and prospective provenance information, and then to interpret and query this information in powerful ways. In particular, our focus on a specific computational models means that we can define a specific schema that maintains information about such constructs as “procedures,” “calls,” and “workflows,” in addition to general purpose assertions. The two schemas can in fact converge: the Moreau schema can subsume the information we describe here, and we can integrate the information of the Moreau schema (in addition to our custom-tailored virtual data schema) by using a more general RDF model for our metadata annotations. With such a schema, metadata annotations can be interpreted broadly, and any annotation can be associated with our core data (file) and executable (procedure, workflow) objects.

PASS [MH+06, BG+06] is a storage system that automatically collects and maintains provenance information. In comparison with VDS, PASS discovers the components and environments required for the production of a specific data item by tracking system calls, where in VDS, we rely on explicit declarations of such dependencies. However, the two systems are complimentary in many ways: PASS can help discover missing pieces in our “declared” provenance, for instance, a data item that is necessary for a derivation process, or an extra data product produced during the process that was not captured in the procedure definition. On the other hand, PASS cannot track complete provenance beyond local file systems, such as in a Grid environment, without provenance-aware applications and environments. Both systems also share similar graph traversal mechanisms to build the ancestry tree for a data product, and have the same set of requirements for provenance and workpattern queries; they also face similar challenges determining the granularity at which the systems should capture provenance information, and the lifetime management of the captured information. PASS has the goal of generalizing the production of certain individual items into a workflow-like pattern, for instance, running “*sort a > b*” would involve the same set of operations for any file produced in such a process; where in VDS we can discover such patterns in an interactive environment such as the Chiron virtual data portal [ZW+05], as users tend to repeat the same derivation process for a set of data items. Similar comments can be made about the automated provenance recording techniques being developed by Barja and Digiampietri within the context of Microsoft’s Windows Workflow Foundation [BD06].

The evolution or provenance of a workflow itself is also vital in scientific analysis. VisTrails [CF+06, FS+06] captures the notion of an evolving dataflow, and implements a history management mechanism to maintain versions of a dataflow, thus allowing a scientist to return to previous steps, apply a dataflow instance to different input data, explore the parameter space of the dataflow, and (while performing these steps) compare the associated visualization results. In VDS, workflows are also composed of individual steps (procedure calls), and chained together by data dependencies. To provide similar functionality, we can either add a versioning mechanism to our system, or maintain workflow construction-specific

metadata annotations to track the history of a workflow. The modification and composition query capabilities described in Section 3 support parameter exploration and comparative analysis. Workpattern queries should allow flexible and powerful workflow editing and transformation.

7 Conclusion and Future Directions

We have described the representation and query of both prospective and retrospective provenance information in our virtual data provenance model, presented examples of how provenance can be employed in representative science processes (in neuroscience and physics), and shown how powerful queries can be used to derive valuable knowledge in data analysis processes. These queries select on various combinations of procedure information, data, metadata, and (what seems particularly interesting) workflow patterns.

While the model described here is based on application programs rather than Web services, we believe the same model of provenance applies equally well in a service oriented architecture, with no loss of generality.

Future extensions to the virtual data provenance model include maintaining a transactional provenance trail of changes to metadata annotations, studies of scalability, management of provenance data retention, and the application of the model to a distributed web of provenance catalogs employing a similar schema (see, for example, AstroDAS [BMP06]). We are also eager to perform more comprehensive usability experiments with a wide range of users.

Acknowledgments

This work was supported in part by the National Science Foundation GriPhyN project under contract ITR-086044 and by the U.S. Department of Energy under contract W-31-109-Eng-38. We acknowledge the contributions to this work of Jens Voekler, Jed Dobson, and members of the QuarkNet project.

References

- [AV+06] Alvarez, S., Vazquez-Salceda, J., Kifor, T., Varga, L.Z. and Willmott, S. Applying Provenance in Distributed Organ Transplant Management, International Provenance and Annotation Workshop (I-PAW), 2006, Springer-Verlag LNCS, 38-47.
- [BG+05] Bardeen, M., Gilbert, E., Jordan, T., Nepywoda, P., Quigg, E., Wilde, M. and Zhao, Y. The QuarkNet/grid collaborative learning e-Lab. IEEE International Symposium on Cluster Computing and the Grid, 2005. CCGrid 2005. Vol. 1 pp. 27-34. 9 May 2005. DOI 10.1109/CCGRID.2005.1558530.
- [BD06] Barga, R.S. and Digiampietri, L.A. Automatic Generation of Workflow Provenance, International Provenance and Annotation Workshop (I-PAW), 2006, Springer-Verlag LNCS, 1-9.
- [BG+06] Braun, U., Garfinkel, S., Holland, D., Muniswamy-Reddy, K. and Seltzer, M. Issues in Automatic Provenance Collection, International Provenance and Annotation Workshop (I-PAW), 2006, Springer-Verlag LNCS, 132-144.

- [BK+06] Bourilkov, D., Khandelwal, V., Kulkarni, A. and Totala, S. Virtual Logbooks and Collaboration in Science and Software Development, International Provenance and Annotation Workshop (I-PAW), 2006, Springer-Verlag LNCS, 19-27.
- [BKT01] Buneman, P., Khanna, S., and Tan. W.-C. Why and Where: A Characterization of Data Provenance. In International Conference on Database Theory, 2001.
- [BM06] Branco, M. and Moreau, L. Enabling provenance on large scale e-Science applications, International Provenance and Annotation Workshop (I-PAW), 2006, Springer-Verlag LNCS, 55-63.
- [BMP06] Bose, R., Mann, R.G. and Prina-Ricotti, D. AstroDAS: Sharing Assertions across Astronomy Catalogues through Distributed Annotation, International Provenance and Annotation Workshop (I-PAW), 2006, Springer-Verlag LNCS, 154-163.
- [CF+06] Callahan, S.P., Freire, J., Santos, E., Scheidegger, C.E., Silva C.T. and Vo, H.T. Managing the Evolution of Dataflows with VisTrails. IEEE Workshop on Workflow and Data Flow for Scientific Applications (SciFlow) 2006.
- [CW00] Cui, Y. and Widom, J., Practical Lineage Tracing in Data Warehouses. In 16th International Conference on Data Engineering, (2000), 367-378.
- [CWW00] Cui, Y., Widom, J. and Wiener, J.L. Tracing the Lineage of View Data in a Warehousing Environment. ACM Transactions on Database Systems, 25 (2). 179-227. 2000.
- [DS+05] Deelman, E., Singh, G., Su, M.-H., Blythe, J., Gil, Y., Kesselman, C., Mehta, G., Vahi, K., Berriman, G.B., Good, J., Laity, A., Jacob, J.C. and Katz, D.S. Pegasus: A Framework for Mapping Complex Scientific Workflows onto Distributed Systems. Scientific Programming, 13 (3). 219-237. 2005.
- [F05] Foster, I., Globus Toolkit Version 4: Software for Service-Oriented Systems. IFIP International Conference on Network and Parallel Computing, 2005, Springer-Verlag LNCS 3779, 2-13.
- [FS+06] Freire, J., Silva, C.T., Callahan, S.P., Santos, E., Scheidegger, C.E. and Vo, H.T. Managing Rapidly-Evolving Scientific Workflows, International Provenance and Annotation Workshop (I-PAW), 2006, Springer-Verlag LNCS, 10-19.
- [FV+02] Foster, I., Voeckler, J., Wilde, M. and Zhao, Y. Chimera: A Virtual Data System for Representing, Querying, and Automating Data Derivation. in 14th Conference on Scientific and Statistical Database Management, (2002).
- [FT+02] Frey, J., Tannenbaum, T., Foster, I., Livny, M. and Tuecke, S. Condor-G: A Computation Management Agent for Multi-Institutional Grids. Cluster Computing, 5 (3). 237-246. 2002.
- [GM+05] Groth, P., Miles, S., Tan, V. and Moreau L. Architecture for Provenance Systems. Technical report, University of Southampton, October 2005.
- [GS02] Giugno, R. and Shasha, D. Graphgrep: A fast and universal method for querying graphs. In Proceeding of the IEEE International Conference in Pattern recognition (ICPR), Quebec, Canada, August 2002.
- [KS06] Kloss, G.K. and Schreiber, A. Provenance Implementation in a Scientific Simulation Environment, International Provenance and Annotation Workshop (I-PAW), 2006, Springer-Verlag LNCS, 28-37.
- [HSM04] Huettel, S., Song, A. and McCarthy, G. Functional Magnetic Resonance Imaging. Sinauer Associates, 2004.
- [MC+03] Myers, J.D., Chappell, A.R., Elder, M., Geist, A. and Schwidder, J. Re-integrating the research record. IEEE Computing in Science & Engineering, pages 44-50, 2003.
- [MH+06] Muniswamy-Reddy, K., Holland, D., Braun, U. and Seltzer, M. Provenance-Aware Storage Systems, 2006 USENIX Annual Technical Conference, Boston, MA, June 2006.

- [SKD06] Singh, G., Kesselman, C. and Deelman, E. Optimizing Grid-Based Workflow Execution. *Journal of Grid Computing*, 3 (3-4). 201-219. 2006.
- [SM03] Szomszor, M. and Moreau, L. Recording and reasoning over data provenance in web and grid services. In *Intl. Conf. on Ontologies, Databases and Applications of Semantics*, LNCS 2888, 2003.
- [WG+98a] Woods, R.P., Grafton, S.T., Holmes, C.J., Cherry, S.R. and Mazziotta, J.C. Automated image registration: I. General methods and intrasubject, intramodality validation. *Journal of Computer Assisted Tomography* 1998;22:139-152.
- [WG+98b] Woods, R.P., Grafton, S.T., Watson, J.D.G, Sicotte, N.L. and Mazziotta, J.C. Automated image registration: II. Intersubject validation of linear and nonlinear models. *Journal of Computer Assisted Tomography* 1998;22:153-165.
- [WS97] Woodruff, A. and Stonebraker, M., Supporting Fine-Grained Data Lineage in a Database Visualization Environment. *13th International Conference on Data Engineering*, 1997, 91-102.
- [ZG+03] Zhao, J., Goble, C., Greenwood, M., Wroe, C. and Stevens, R. Annotating, linking and browsing provenance logs for e-science. In *Workshop on Semantic Web Technologies for Searching and Retrieving Scientific Data*, October 2003.
- [ZGR06] Zhao, J. Goble, C. and Stevens, R. An Identity Crisis in the Life Sciences, *International Provenance and Annotation Workshop (I-PAW)*, 2006, Springer-Verlag LNCS.
- [ZD+05] Zhao, Y., Dobson, J., Foster, I., Moreau, L. and Wilde, M. A Notation and System for Expressing and Executing Cleanly Typed Workflows on Messy Scientific Data. *SIGMOD Record*, 34 (3). 37-43. 2005.
- [ZW+05] Zhao, Y., Wilde, M., Foster, I., Voeckler, J., Dobson, J., Gilbert, E., Jordan, T. and Quigg, E. Virtual Data Grid Middleware Services for Data-Intensive Science. *Concurrency and Computation: Practice and Experience*, DOI: 10.1002/cpe.968, 2005.

A Provenance Model for Manually Curated Data

Peter Buneman¹, Adriane Chapman², James Cheney¹,
and Stijn Vansummeren^{3,*}

¹ University of Edinburgh

² University of Michigan, Ann Arbor

³ Hasselt University and Transnational University of Limburg, Belgium

Abstract. Many curated databases are constructed by scientists integrating various existing data sources “by hand”, that is, by manually entering or copying data from other sources. Capturing provenance in such an environment is a challenging problem, requiring a good model of the process of curation. Existing models of provenance focus on queries/views in databases or computations on the Grid, not updates of databases or Web sites. In this paper we motivate and present a simple model of provenance for manually curated databases and discuss ongoing and future work.

1 Introduction

Many scientific databases¹ are constructed by manual effort of scientists acting as *curators*. Curators use a wide variety of sources to select, organize, classify and annotate existing data into a database on some topic. Such databases are now supplanting printed “reference manuals” as standard sources of raw scientific data. Although they are most widespread in bioinformatics, scientific databases in many disciplines involve a degree of manual curation.

Curation typically involves “manually” reading journal articles or browsing remote databases to find relevant new information. Data gleaned from journal abstracts or copied from other databases is entered directly by the curator using a Web form or custom interface (such as a MS Access application). This manual process, in which the scientist’s brain is in the inner loop, is what distinguishes curation from related activities such as data integration. Curated data is generally of higher quality, but is correspondingly more expensive to produce.

The perceived value of curated databases rests on their provenance: that is, the fact that they have been constructed by well-informed individuals who have exercised scientific judgment in assimilating data sources. However, the volatile nature of electronic media makes it difficult to trust blanket assertions about the provenance of such databases. Instead, it is widely believed that explicit evidence about the provenance of such databases must be recorded in order to

* Postdoctoral researcher of the Fund for Scientific Research - Flanders.

¹ In this paper the term “database” should be interpreted broadly to include data stored in flat files, XML documents, Web sites, etc., not just RDBMSs.

preserve the scientific record and assess the scientific value of such databases (and any results derived from them). This *provenance information* should, at least, indicate the origin, context, intermediate source(s) and modification history of the data. Besides its intrinsic value as part of the scientific record, provenance can be used to detect duplicate copies of information and to help automate “data cleaning” tasks such as propagating corrections to source databases and tracking down the sources of discrepancies.

While it is very easy to build up a database by searching for, copying and pasting data, maintaining provenance information adequate for scientific applications requires additional effort, which adds to the already-high cost of curated data. Curators usually attempt to add links to the original publications or source databases, but in practice, provenance records are often absent, incomplete or ad hoc, often despite curators’ best efforts. Also, manually-managed provenance records are at higher risk of human error or falsification. Since a great deal of information relevant to provenance is available to the systems involved, we believe that it is important to develop techniques for data curation which automate provenance management as much as possible.

Previous approaches to provenance management [1,6,7,9,11,18] typically have focused on situations in which all of the interactions with data take place in a single controlled environment (e.g. an operating system, file system, or database), or in which new data is only constructed from existing data using nondestructive mechanisms such as database views or scientific workflows. Both assumptions are violated in the case of manually curated databases. Tracking the provenance of data that moves among databases or Grid resources is challenging because there is no one system that can capture all of the actions involved; instead, many systems must cooperate in order to maintain the chain of provenance. Also, curated databases are updated in-place with local copies of source data rather than constructed as views of source databases.

The purpose of this paper is to describe the challenges involved in managing provenance for manually curated databases, and to summarize our current approach to them. Section 2 defines the problem and describes the constraints which we believe must be met by a practical solution. In Section 3, we propose a *copy-paste model* for describing user actions in assimilating external data sources into curated database records. We define provenance as a relation between versions of a database describing how each part of the output was derived from data in earlier versions or external sources. Section 4 discusses ongoing and future work on implementing and extending the copy-paste model, including a proof-of-concept implementation, approaches to tracking user actions, extensions to the copy-paste model, and data citation.

2 The Problem

Biological databases such as UniProt [16] and the Nuclear Protein Database [8] are manually curated; that is, they are constructed (at least in part) by curators modifying individual records directly rather than by an automatic view

or data extraction process. To the extent that extraction, cleaning and integration operations involve user interaction, data warehouses [7] could also be considered to be manually curated. For the moment we focus on tracking provenance for fine-grained, manual updates, rather than queries, views, or “bulk” updates.

We define the *provenance management problem* for manually curated databases as follows. Given a definition of the complete and correct history of a database as it evolves over time, the goal is to store sufficient provenance information to be able to answer queries about the history given only the provenance information and the final database state(s). Note that we do not assume an absolute definition of history; instead, the appropriate form of historical information depends on the application.

This provenance management problem may seem already solved; for example, file systems, database triggers, version control systems, and Wikis provide adequate provenance management (in the form of creation/modification timestamps, user activity logs, change logs, etc.) for their application areas. However, all of these systems rely on strong assumptions: there is a single system that monitors all access to the data, and the data is stored in a single format. When information crosses boundaries between systems, such provenance information usually becomes invalid, and there is no way to say that data in one system comes from another system (possibly with a different data model). Similarly, solutions using Grid technology [12] and customized e-notebook or workbench software [15] require a substantial level of coordination among databases and applications. While such tools are likely to be beneficial, it will take time for them to become widely adopted among scientists. Recently, Muniswamy-Reddy et al. [3,13] have developed a “provenance-aware storage system” (PASS), which tracks provenance at the file system level. While certainly relevant (especially for scientific data stored using files), PASS addresses provenance issues orthogonal to those arising in other forms of databases.

In contrast to common data integration situations in business, there is no central authority that controls all of the scientific databases relevant to a given area. Thus, no single system can monitor all scientific databases or mandate changes or standard practices. In addition, not all databases are being actively maintained, and others may be resistant to change. Even motivated database curators may lack the resources to modify their own databases. Conversely, databases often change independently and have widely varying record-keeping practices, use a wide variety of data models (ranging from RDBMS to flat files to XML to filesystems), and their curators employ a wide variety of independent tools (Web browsers, editors, database access applications, etc.)

Under these constraints, we believe that a practical solution must have the following characteristics:

- **decentralized**: deployable one database at a time, without requiring cooperation among all databases at once
- **data model-independent**: should work for data stored in flat file, relational, XML, file system, Web site, etc. models

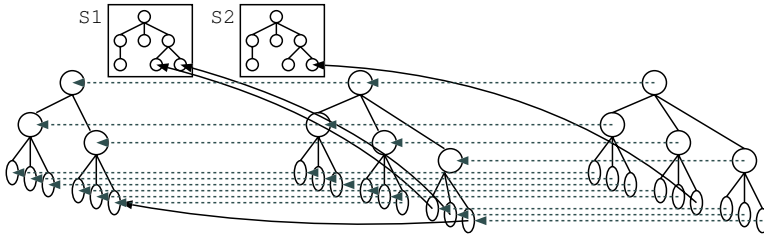


Fig. 1. A two-step provenance record

- **usable with minimum changes to existing curatorial practice:** ideally, provenance tracking is invisible to the user
- **useful without significant changes to existing database systems:** since it is impractical to impose global standards across databases
- **robust in the face of changes to the databases:** since we cannot stop other databases from changing anyway
- **scales gracefully** to situations in which many databases cooperate to maintain provenance chains.

3 The Copy-Paste Model

In this paper, we focus on the problem of managing provenance for a single database as data is copied from external sources or modified within the target database. This is a common situation, and we believe our approach meets most of the criteria above; some issues, such as robustness in the face of changes to other databases, are not yet handled.

As remarked by Groth et al. [10], it is important to think carefully about what provenance information to retain, based on what information will be useful evidence to later observers. In our setting, we aim only to track the internal relationships among a sequence of versions of the target database and fixed source databases. Thus, we define the “complete and correct history” in the above problem statement as follows. We take an abstract view of databases as maps from data locations (or *citations*) to data values. For the purposes of this paper, we consider only a “flat” data model, in which a database is simply a set of key-value pairs. Many other refinements are possible, for example using line numbers to locate data in text files, XPath or XPointer expressions to address data in XML, or using key information to locate data in relational databases. A more realistic instance of our model that uses paths to address relational or XML data can be found in [5].

We further assume that changes to the database can be modeled using transactions comprising simple “copy-paste” updates. Updates (denoted by U) are sequences of atomic editing operations: insertion $\text{ins } l \ v$ which creates a new location l that is mapped to data value v , deletion $\text{del } l$ which deletes location l (and its corresponding data value); and copy-paste $l \leftarrow l'$ which updates the value at

location l to the value mapped by l' . We write $I \xrightarrow{U} O$ to for a triple (I, U, O) such that applying update U on input database I yields output database O . The *one-step provenance relation* $\mathcal{P}(I \xrightarrow{U} O)$ is a subset of $\text{dom}(I) \times \text{dom}(O)$, defined as follows:

$$\begin{aligned}\mathcal{P}(I \xrightarrow{l \leftarrow l'} O) &= \{(m, m) \mid m \in \text{dom}(I), m \neq l\} \cup \{(l', l)\} \\ \mathcal{P}(I \xrightarrow{\text{ins } l^v} O) &= \{(m, m) \mid m \in \text{dom}(I)\} \cup \{(\text{NULL}, l)\} \\ \mathcal{P}(I \xrightarrow{\text{del } l} O) &= \{(m, m) \mid m \in \text{dom}(I) - l\} \cup \{(l, \text{NULL})\} \\ \mathcal{P}(I \xrightarrow{U; U'} O) &= \bigcup \{r \otimes s \mid r \in \mathcal{P}(I \xrightarrow{U} J), s \in \mathcal{P}(J \xrightarrow{U'} O)\}\end{aligned}$$

where $r \otimes s$ is defined as follows:

$$\begin{aligned}(l, m) \otimes (m, n) &= \{(l, n)\} && (l, m, n \neq \text{NULL}) \\ (l, m) \otimes (m, n) &= \{(\text{NULL}, l)\} && (l = \text{NULL} \text{ or } m = \text{NULL}) \\ (l, m) \otimes (m, n) &= \{(l, \text{NULL})\} && (m = \text{NULL} \text{ or } n = \text{NULL}) \\ r \otimes s &= \emptyset && \text{otherwise}\end{aligned}$$

Given a sequence of database instances and updates $I_0 \xrightarrow{U_1} \dots \xrightarrow{U_n} I_n$, we define the multi-step provenance relation $\text{Prov}(\text{Sid}, \text{Loc}, \text{Loc})$ as

$$\{1\} \times \mathcal{P}(I_0 \xrightarrow{U_1} I_1) \cup \dots \cup \{n\} \times \mathcal{P}(I_{n-1} \xrightarrow{U_n} I_n).$$

Intuitively, $\mathcal{P}(I \xrightarrow{U} O)$ (and consequently, Prov) describe how data is copied, inserted, or deleted during a transaction (or sequence of transactions). Figure 1 depicts an example two-step provenance relation on a database with tree-structured locations. Here, the “boxed” trees S_1, S_2 are external sources, whereas the other trees denote the original database, the database after the first update, and the database after the second update, respectively. The solid and dashed lines describe the Prov relation; dashed lines indicate provenance links for *unchanged* data. Such links can always be omitted from the Prov relation to save space.

Using the Prov relation, we can define a number of interesting basic provenance queries giving information about the creation time of a location, the deletion time, where a location was copied from, and whether a location has been modified:

$$\begin{aligned}\text{Inserted}(t, l) &\leftarrow \text{Prov}(t, \text{NULL}, l). && \text{Deleted}(t, l) \leftarrow \text{Prov}(t, l, \text{NULL}). \\ \text{Copied}(t, l, m) &\leftarrow \text{Prov}(t, l, m), l \neq m. && \text{Unchanged}(t, l) \leftarrow \text{Prov}(t, l, l).\end{aligned}$$

The $\text{Inserted}(t, l), \text{Copied}(t, l', l), \text{Unchanged}(t, l)$ queries informally say that the data at location l at the end of transaction t was inserted during t , copied from the data at l at the beginning of t , or unchanged during t , respectively; similarly, $\text{Deleted}(t, l)$ says that the data at l was deleted during t .

Transaction identifiers can be used to index a table $\text{Trans}(\underline{\text{Tid}}, \text{Uid}, \text{Time}, \dots)$ containing additional metadata about transactions. The following recursive query $Q(l, \text{tid}, \text{uid}, t)$ defines the relation “the data at l at the end of transaction tid was originally inserted by user uid at time t ”:

$$\begin{aligned}
Q(l, tid, uid, t) &\leftarrow \text{Inserted}(tid, l), \text{Trans}(tid, uid, t). \\
Q(l, tid, uid, t) &\leftarrow \text{Copied}(tid, l, m), Q(m, tid - 1, uid, t). \\
Q(l, tid, uid, t) &\leftarrow \text{Unchanged}(tid, l), Q(l, tid - 1, uid, t).
\end{aligned}$$

Many interesting provenance queries also refer to the raw data. A simple example is to return the original source (that is, the external link, or NULL if none) alongside each result. Another example is using provenance to filter out data from known unreliable sources, or to group or aggregate the data by source. Querying the data and its provenance “side-by-side” involves additional processing, since the data and provenance may be stored in separate databases. However, existing techniques for multidatabase querying and managing annotations can be used for this purpose [1,17].

If only one database tracks provenance, then the chain of provenance can only be followed to the point where data was copied from an external source; only “local” provenance queries can be answered. We can only answer “global” queries if all the databases involved record provenance. Maintaining consistent and valid provenance when the distributed databases are being updated asynchronously is an interesting area for further research.

4 Ongoing and Future Work

4.1 An implementation

We have implemented a “copy-paste database”, CPDB, that tracks the provenance of data copied from external sources to the target database [5]. CPDB permits the user to copy source data from external sites or databases into the target database, and modify the data to fit the target database’s structure. The user’s actions are intercepted and provenance information is recorded in a *provenance store*. CPDB uses paths as locations to address data stored in either XML or relational form. CPDB addresses a number of implementation issues that were left implicit in the discussion in Section 3. We now discuss these issues further.

How Can We Address and Update Heterogeneous Data? The approach we take to the first issue is to require that the citable data in every source database is published as a “fully-keyed” XML view, and the updatable data in the target database is presented as an *updatable* XML view. Thus, paths can be used to select data from sources and locate data in copy-paste updates to the target database. We believe these are reasonable requirements since XML publishing and view update for legacy RDBMSs are widely recognized as important research problems and have already received attention [2]. Moreover, many commercial relational databases already provide some capability for publishing their data as XML. In addition, any Web page can be interpreted as an XML document, addressed by its URL and an XPath or XPointer describing the cited data.

This approach does *not* require that any of the source or target databases represent data internally as XML. Any underlying data model for which path addresses make sense can be used. Also, the databases need not expose all of their data. Instead, it is up to the databases' administrators how much data to expose for copying or updating. The data in many scientific databases consists of a "catalog" relation that contains all the raw data, together with supporting cross-reference tables. It is only this catalog that would typically need to be made available by a source database.

Proxies or client-side scripting could also be used to add more useful citations to non-cooperating database websites without changing the curator's natural behavior. However, a chain is only as strong as its weakest link, and the provenance data derived from this method can only contain information about the database's website, not any of the underlying database values. This is related to the problem of *data citation* discussed below.

How Can We Capture the User's Actions in an Unobtrusive Way?

The current implementation of CPDB provides a Web interface that enables the user to import data from a source database and paste it into the target. However, this interface is relatively clumsy compared to what curators normally do, namely browsing to databases' Web sites, copying data from the relevant pages, and pasting it into a target database entry form. For an approach to automatic provenance tracking to be successful, it has to make life easier for the curators, not harder.

Fortunately, it appears to be possible to instrument Web browsers so that the user's browsing, copying, pasting, and form submission actions are recorded as provenance records. We are currently investigating whether existing techniques for Web browser activity logging being developed in the human-computer interaction community [14] can be adapted to record rich provenance information unobtrusively.

How Can the Resulting Provenance Records be Stored and Queried Efficiently?

The *Prov* relation given in the previous section is potentially very verbose since it stores links between unchanged data in subsequent versions. (These are shown by the dotted lines in Figures 1). Thus, the number of links between two versions of a database is proportional to the size of the data, not the difference between the versions. While these links are needed to answer provenance queries, they do not need to be stored explicitly. In CPDB, we store only "essential" provenance links, that is, links having to do with copies, inserts, or deletes. CPDB also exploits optimization opportunities offered by the tree structure of path addresses. The full *Prov* relation can then be defined as a view of the reduced form. The size of the reduced form of *Prov* is proportional only to the size of the update operation, not the size of the database.

This reduces the amount of storage space needed for the provenance of manual updates to an acceptable level. We also performed experiments that show that the performance of queries to the core provenance relation is better than for

more explicit forms of *Prov*. The reason for this is that it is often faster to calculate provenance links “on the fly” than to fetch them from storage.

4.2 Future Work

Database Archiving and Citation. The ability to locate data precisely in the source database is fundamental to the copy-paste model we developed in Section 3. It is also important if the curator has not directly copied the data, but has interpreted it and wants to provide a citation to that portion of the database. However, in addition to location information, citations carry other useful data such as authorship, title, etc. It is becoming increasingly important to make curated databases citable and to describe how a database or a part of it should be cited [4]. We are currently considering techniques for augmenting database archiving and citation with support for provenance tracking and querying.

Extending Our Model of Provenance. The copy-paste language introduced in Section 3 suffices to model the behavior of a human curator who inserts individual items into the curated database. Ideally, however, we would also like to track the provenance of data constructed by automatic processes (e.g., a database query, a workflow, a scientific algorithm, ...). We are currently investigating extensions of our approach to provenance to full query/update languages for relational, complex object, and XML data, including aggregation and “fusion” operations such as summation, joins or unions. Among the research challenges here are finding space-efficient, compact representations for such provenance information and providing appropriate high-level query language operations for querying more complex provenance expressions.

In addition, we are generalizing the copy-paste model to study the provenance behavior of arbitrary computations based on *rewriting machines*, a variant of Turing machines. In Turing machines, there is at best an implicit correlation between input and output data. Without knowing the intent of the designer of a machine, it is impossible to give a correct provenance semantics for it. In rewriting machines, the basic machine operations are *rewritings*, which generalize copy-paste operations and have a natural, unambiguous provenance semantics.

5 Conclusions

Tracking the provenance of data in manually curated databases is challenging, primarily because the most obvious approaches require unrealistic levels of homogeneity, cooperation and coordination among databases. While the long-term solution may require changes to common scientific practice, there are steps that can be taken in the short term and at the local level to improve record-keeping in current practices. We have developed and implemented a model for recording the provenance of data that flows into a single curated database and discussed incremental extensions to this model that provide further improvements. Many substantial challenges remain.

References

1. D. Bhagwat, L. Chiticariu, W. C. Tan, and G. Vijayvargiya. An annotation management system for relational databases. In *Proc. of the Intl. Conf. on Very Large Data Bases (VLDB)*, pages 900–911. Morgan Kaufmann, 2004.
2. V. P. Braganholo, S. B. Davidson, and C. A. Heuser. From XML view updates to relational view updates: old solutions to a new problem. In *VLDB 2004*, pages 276–287, 2004.
3. U. Braun, S. Garfinkel, D. A. Holland, K.-K. Muniswamy-Reddy, and M. I. Seltzer. Issues in automatic provenance collection. In *Proceedings of the 2006 International Provenance and Annotation Workshop (IPAW 2006)*, 2006. This proceedings.
4. P. Buneman. How to cite curated databases and how to make them citable. In *SSDBM*, 2006. To appear.
5. P. Buneman, A. P. Chapman, and J. Cheney. Provenance management in curated databases. In *SIGMOD*, 2006. To appear.
6. P. Buneman, S. Khanna, and W.-C. Tan. Why and Where: A characterization of data provenance. In *ICDT*, pages 316–330, 2001.
7. Y. Cui and J. Widom. Lineage tracing for general data warehouse transformations. In *Proceedings of the 27th VLDB Conference, Roma, Italy*, pages 41–58, 2001.
8. G. Dellaire, R. Farrall, and W. A. Bickmore. The nuclear protein database (NPD): sub-nuclear localisation and functional annotation of the nuclear proteome. *Nucleic Acids Research*, 31(1):328–330, 2003.
9. I. Foster, J. Vockler, M. Eilde, and Y. Zhao. Chimera: A virtual data system for representing, querying, and automating data derivation. In *International Conference on Scientific and Statistical Database Management*, pages 1–10, July 2002.
10. P. Groth, S. Miles, , and S. Munroe. Principles of high quality documentation for provenance: A philosophical discussion. In *Proceedings of the 2006 International Provenance and Annotation Workshop (IPAW 2006)*, 2006. This proceedings.
11. P. Groth, S. Miles, W. Fang, S. C. Wong, K.-P. Zauner, and L. Moreau. Recording and using provenance in a protein compressibility experiment. In *HPDC*, 2005.
12. P. T. Groth, M. Luck, and L. Moreau. A protocol for recording provenance in service-oriented grids. In T. Higashino, editor, *OPODIS*, volume 3544 of *Lecture Notes in Computer Science*, pages 124–139. Springer, 2004.
13. K. Muniswamy-Reddy, D. Holland, U. Braun, and M. Seltzer. Provenance-aware storage systems. In *Proceedings of the 2006 USENIX Annual Technical Conference*, Boston, MA, June 2006. To appear.
14. N. Roussel, A. Tabard, and C. Letondal. All you need is log. WWW2006 Workshop on Logging Traces of Web Activity: The Mechanics of Data Collection, May 2006. Manuscript available at <http://torch.cs.dal.ca/~www2006/roussel-www2006-MechanicsDataCollection.pdf>.
15. R. D. Stevens, A. J. Robinson, and C. A. Goble. *myGrid*: personalised bioinformatics on the information grid. *Bioinformatics*, 2003.
16. UniProt. <http://www.ebi.ac.uk/uniprot/>.
17. Y. R. Wang and S. E. Madnick. A polygen model for heterogeneous database systems: The source tagging perspective. In D. McLeod, R. Sacks-Davis, and H.-J. Schek, editors, *16th International Conference on Very Large Data Bases, August 13-16, 1990, Brisbane, Queensland, Australia, Proceedings*, pages 519–538. Morgan Kaufmann, 1990.
18. J. Widom. Trio: A system for integrated management of data, accuracy, and lineage. In *CIDR*, pages 262–276, 2005.

Issues in Automatic Provenance Collection

Uri Braun, Simson Garfinkel, David A. Holland,
Kiran-Kumar Muniswamy-Reddy, and Margo I. Seltzer

Harvard University, Cambridge, Massachusetts
`pass@eecs.harvard.edu`

Abstract. Automatic provenance collection describes systems that observe processes and data transformations inferring, collecting, and maintaining provenance about them. Automatic collection is a powerful tool for analysis of objects and processes, providing a level of transparency and pervasiveness not found in more conventional provenance systems. Unfortunately, automatic collection is also difficult. We discuss the challenges we encountered and the issues we exposed as we developed an automatic provenance collector that runs at the operating system level.

1 Introduction

Today's provenance management systems usually take one of two approaches to provenance collection: Either users enter it manually or applications explicitly collect provenance and enter it into a database. There is, however, a third model: *automatic provenance collection*. In automatic collection, the system observes the actions of users and programs and derives provenance, storing it without user or application involvement.

Automatic collection is a powerful approach, because it eliminates user error, consistently collects provenance across all applications, and captures more complete provenance than systems relying on a user's or application developer's assumptions about provenance. For example, automatic provenance collection in the operating system allows us to: identify system configuration changes (*e.g.*, new tools or libraries), identify environment variable modifications that alter program behavior, debug faulty builds that are missing dependencies, identify the source and creation of unusual files, and create scripts that produce objects.

In earlier work we described a prototype Provenance-Aware Storage System (PASS), built on Linux, that automatically collects provenance at the operating system level [18]. PASS is similar to systems such as ClearCase [5], GenePattern [9], and Vesta [10]. These systems observe users' and applications' activities, recording the provenance captured in these activities. PASS takes this one step farther, observing *all* processes that run on a PASS-enabled operating system, generating provenance for objects that do not have provenance (*i.e.*, are *unprovenanced*), and attaching complete system-level provenance to objects that are created on a provenance-aware file system. We capture low-level details like the operating system, kernel modules loaded, installed libraries, and process environment.

We found that automatic OS-level provenance collection is useful, complementing existing approaches. However, we exposed a number of challenging issues that arise from automatic collection. This paper introduces automatic provenance collection and discusses the more interesting challenges that arise in building these systems.

In Section 2 we define automatic provenance collection, placing it in the context of existing provenance solutions. In Section 3 we discuss the problems that arise when designing and building systems with automatic provenance collection. In Section 4, we present use cases where disclosed and observed provenance systems together provide more powerful solutions than either independently. In Section 5 we introduce *provenance pruning*, the deletion of provenance, and discuss strategies for implementing it. In Section 6 we discuss the privacy implications of automatic provenance collection. In Section 7 we discuss automatic provenance systems and technologies from which we can borrow in constructing automatic provenance systems, and in Section 8 we conclude.

2 What Is Automatic Collection?

An automatic collecting system transparently records provenance for *all* activities it performs by observing the sequence of operations executed and translating relevant ones into provenance. For example, when a process begins running, the operating system identifies several process provenance attributes ¹, such as the executable, operating system, loaded kernel modules, libraries, environment, and command line.

The system continues to collect provenance about the running process, recording (for example) input sources. Whenever the process creates or modifies an object, the process's provenance is assigned to the written object. We call this form of provenance collection *observed*, because the system derives provenance from the events that it observes. An observed provenance system does not necessarily understand the semantics of its observations, so it must record everything that is potentially part of an object's provenance. This can lead to *false provenance* if the observing system does not perform detailed information flow analysis. Section 2.1 discusses existing systems that use observed provenance.

Most existing provenance systems use *disclosed* provenance. In disclosed provenance systems, users or applications present provenance to the system, using the provenance system merely as a storage and query engine. There are several kinds of disclosed provenance. *Manual* provenance, sometimes called *annotation*, is entered by users. For example, the provenance of data entered manually by a user must itself be manually entered. *Specified* provenance describes an object's intended provenance in a structured way, typically by directing the system to produce the object via various transformations or workflows. For examples, workflow-based systems [23,32] and makefiles capture intended provenance by describing how a target is created from its sources. The workflow systems

¹ We think of processes as having provenance so that we can transfer a process's provenance to objects it creates.

frequently then generate true provenance from the execution of these workflows, while `make` does not. Instead, the canonical software development environment relies on a separate component, a source code control system, to store semantic provenance. In these systems, the difference between successive versions precisely identifies *what* changed, but the rationale is entered as manual provenance in the form of commit messages.

2.1 Observed-Provenance Systems

There are several domain-specific observed provenance systems. GenePattern [9] is a working environment for computational biology and biomedical research. It tracks provenance for the objects created in the environment. Clearcase [5] and Vesta [10] are environments designed for software development. Like our provenance-aware storage system (PASS), both Clearcase and Vesta use a customized file system to track provenance. Unlike PASS, both Clearcase and Vesta center their design on disclosed provenance, assuming that users execute commands specified in a control file. However, both allow some form of observation in which the system observes a user's actions creating a control file (script) to re-derive objects. Running in this observed mode is not the norm and requires explicit action by the user.

Observed provenance systems provide the benefit that provenance collection is automatic, requiring no user intervention. Observed systems collect and maintain provenance of objects by default, so *all* new data becomes provenanced. The major disadvantage of observed provenance systems is that they can only capture provenance to which they are exposed, and this frequently produces provenance with less semantic meaning than disclosed provenance systems. The next section illustrates how disclosed provenance systems complement observed provenance systems.

2.2 Disclosed-Provenance Systems

The vast majority of today's provenance systems use disclosed provenance. These systems require that the user describe a workflow, and then they provide an engine that executes the workflow. The combination of the workflow specification and the result created by running the engine on the workflow creates provenance. The line between observed and disclosed systems is not always crisp. In workflow-specification systems, the actual workflow engine is an observed provenance system; however, it is an observed system that relies on an underlying specification in order to function. Without the specification, the workflow engine cannot generate provenance. For expository purposes, we classify these systems as disclosed provenance systems, because they require the user to disclose explicitly the intended provenance of a result.

The myGrid [32] workflow enactment engine records provenance for each step in a workflow, including inputs and outputs, storing the provenance in a central provenance repository. The PASOA [23] project provides APIs that clients and services use to record provenance during workflow execution. Chimera [7]

offers a virtual data system that provides a virtual data language (VDL) and a virtual data catalog (VDC). The VDC implements a virtual data schema that defines the objects and relations that can be used to capture descriptions of program invocations and to record potential or actual invocations. The Earth System Science Workbench (ESSW) [8] is a data management infrastructure used for processing satellite imagery. The CMCS (Collaboratory for the Multi-scale Chemical Sciences) [21] uses a portal and metadata-aware content store as a base for building a system to support inter-domain knowledge exchange in chemical science. All these systems rely on some sort of specification or explicit API for provenance disclosure. Disclosed provenance systems also make use of user annotations. In fact, some systems [29] rely mostly on user-provided annotations.

Disclosed provenance systems usually provide richer semantic knowledge than observed systems. However, to obtain this benefit, users are restricted to using provenance-aware tools and must conduct their work within the confines of applications that understand how to collect provenance. Thus, the responsibility for provenance collection is dispersed throughout many different tools.

2.3 Observed Provenance Complements Disclosed Provenance

Observed and disclosed provenance provide complementary solutions. An ideal solution provides the full semantic knowledge of disclosed provenance using the automatic and transparent collection of observed provenance systems. While there has been significant research on disclosed provenance systems, there has been significantly less on observed provenance systems. To build systems with the benefits of both approaches, we must understand and overcome the challenges that automatic collection presents. We next discuss some of those challenges, and then in Section 4, we return to examples of how observed and disclosed systems complement one another.

3 Challenges

An observed provenance system faces the challenge of transforming observations into disclosed provenance. This transformation requires solving several problems. There may be several types of mismatches between the observed and desired provenance. We use the term *granularity* to refer to the mismatch between the operating system's observation of a sequence of system calls and the scientific user's desire to record provenance on an experiment.

Reconciling these mismatches leads to challenges in creating and maintaining provenance ancestry; automatic provenance collection can lead to cyclic ancestry, which is conceptually unacceptable. Versioning is one way to cope with cyclic ancestry, but it presents challenges similar to those found in automatically versioned file systems [25]. In the rest of this section, we explore each of these issues in more detail.

3.1 Granularity

Granularity refers to the types of objects for which a system maintains provenance. Coarse grain provenance might describe data and results produced by an entire research initiative (e.g., the Human Genome Project). At the other extreme, the programming languages and systems communities are sometimes interested in extraordinarily fine-grained provenance, such as byte- or bit-level provenance [17,26].

Users are most frequently interested in coarse-grained provenance, such as an experiment or analysis. However, automatic collection systems most naturally operate at a finer grain.

Our prototype PASS collects system call events, recording provenance on a per-file basis. This is the most natural model for an operating-system-based provenance collector, but we can increase the utility of our system by creating coarse-grained views that are interesting to users. If the coarse-grain view is the only interesting view, it is best to perform this conversion (from fine-grain to coarse-grain) at collection time, reducing the provenance overhead.

Automatically identifying and constructing these coarser-grain views remains an open research problem for both data and the events that produce data. Our PASS prototype supports user annotations, which are a manual way to provide coarser views. We are exploring other approaches such as describing classes of files for which provenance is unnecessary (e.g., temporary files), in which case the output files of interest will have the provenance of the entire transformation. Identifying meaningful events (i.e., event granularity) poses a much larger problem, as it is tightly coupled to versioning.

3.2 Versioning

A system that both tracks provenance and allows data modification is inherently versioned. Each modification to a provenanced object changes the provenance of the object and creates a new version of that object, regardless of whether the system actually retains those different versions. On a system that does not explicitly track such versions, provenance provides the connections between versions of objects and distinguishes those objects at different points in their lifetimes.

Like an automatic versioning file system, a provenance-aware system must decide what constitutes a “modification” and thus when to declare new versions. The simplest approach, used by Wayback [4], creates a version on every write call. This simple approach is impractical for automatic provenance collection, because it yields too many meaningless versions. Another method, used in versioning file systems [15,25], is copy-on-write, where a new version is created on the first write to a file between an open and close. A slightly different approach, copy-on-change [12], creates new versions only if a write actually modifies the object. A third alternative, used in a number of commercial and research systems [11,13,22,24], is to take snapshots, or checkpoints. In this model, a snapshot contains the version of each file that existed when the snapshot was taken. These systems typically take snapshots at regular intervals rather than being triggered

by system activity. In PASS, we do copy-on-write: we consider a version complete and “frozen” on the last close (or on `fsync`) and on the next write a new version is created. A new version is also created if a file is truncated to zero length.

Copy-on-write and copy-on-change both involve grouping related sets of modifications into a single new version. This is a form of event granularity abstraction: combining multiple observed write operations into a single conceptual write operation. This merging process requires extreme caution in automatic provenance collection, because it can lead to an even more vexing problem: cycles.

3.3 Cycles

Any modification grouping mechanism introduces the possibility of cycles in the provenance. Cycles in provenance are nonsensical; an object cannot be descended from itself. Cycles may even violate causality: an object may not be created by another object it had somehow previously emitted. So automatic systems must avoid creating provenance cycles. Unfortunately, avoidance is difficult.

Consider the common behavior of a program that first reads and then writes a file. Either the write creates a new version of an object or it creates a provenance cycle. This simple case is easy to resolve, but suppose this process repeats in a loop. The provenance collector cannot observe the loop, only the read and write actions; to avoid creating multiple versions it must infer the existence of the loop and take appropriate steps.

Inferring loops in a single process is tractable, but systems of interacting processes can also produce cycles. Local knowledge is not necessarily sufficient to identify the situation, much less correct it. Consider two processes, P and Q:

P	Q
read a	
	read b
write b	
	write a

If no new versions are created, these processes produce a cycle, yet nothing about P or Q in isolation makes that evident. Simply creating additional versions does not adequately solve the problem; cyclic workloads produce too many versions. (For example, consider a parallel, iterative algorithm.) The solution is to abstract P and Q into a single higher-level conceptual entity, making the cyclic data flow internal to this higher-level entity. Internal data flow need not be provenanced.

Our prototype PASS maintains a global relationship graph and checks it for cycles every time an edge is added. This allows it to create higher-level abstractions. Our current algorithm creates a new version of every process involved in a cycle and then merges these versions together. The newly created merged object becomes the parent of all the files involved, without creating new versions. This handles many common cases with minimal overhead, but fails in some circumstances.

The cycle problem is inherent in any attempt to reduce the number of versions generated and thus inherent in event granularity abstraction. The same problem can and will appear in any automatic collection system. These problems do not appear in disclosed provenance systems, because statements of disclosed provenance are initiated by a human developer, who understands the granularity of the operations. In observed systems, we must deduce the granularity to identify the semantically correct grouping.

Cycle detection and elimination has been a major preoccupation of our recent research; nonetheless, we still do not have a satisfactory algorithm; it remains an open problem [2].

4 Integrating Observed and Disclosed Provenance

Systems that support both observed and disclosed provenance offer powerful features that cannot be obtained using either type alone.

Consider running the GenePattern [9] system on top of PASS. GenePattern possesses the semantic knowledge to record the exact analysis used to transform input data to output results. However, it does not know what version of the math library or what floating point processor was used. This information may be available to GenePattern, but not always, nor in a portable or reliable fashion. In contrast, PASS is integrated with the operating system, handling such issues natively. Together, GenePattern and PASS can answer the query, “Why did this identical transformation produce different results last week and this week?”

Alternatively, suppose that outside of the GenePattern environment a researcher “fixed” an input file. No provenance query in GenePattern can detect that the input file changed, explaining how identical analyses on “the same” input file yield different results.

As discussed previously, a significant challenge for observed provenance systems is identifying a semantically meaningful level of granularity. When a disclosed provenance system sits atop PASS, that disclosed provenance system provides precisely the information PASS needs to construct these coarser views.

5 Pruning

Provenance adds storage overhead. *Provenance Pruning* is the act of selectively removing provenance to save space. In general, pruning removes the provenance for entire objects; however, at the end of this section we consider approaches that erase only part of an object’s provenance.

Storage overhead can grow rapidly. Left unchecked there is nothing to prevent the provenance from dwarfing the data it describes. Such large space overhead can also harm query performance. For example, in recent work [18] we showed that small changes to a source file in the Linux kernel generated approximately two kilobytes of additional provenance when the kernel was rebuilt.

Some provenance can be pruned immediately at collection time. Users may not be interested in recording configuration files, such as `.bashrc` and `.profile`,

as ancestors of the bash shell. Similarly, users may not be interested in some output files, such as temporary files or `/dev/null`. In other cases, it would be useful to identify entire processes and the objects they modify as not requiring provenance. For example, `makewhatis`, which indexes man pages, examines all of them before writing out the index file; the provenance of that index file is thus quite voluminous and also completely uninteresting. Such specification of unprovenanced objects allows the system to prune provenance before it is ever recorded to disk.

There are also opportunities for provenance pruning after collection. Deleted files without descendants are good candidates for pruning at any time. It is useful to think of provenance as forming a tree where parent nodes are those nodes accessed as input during the creation of their children. In such a tree, the deletion of a leaf node can be accompanied by the deletion of that node's provenance, since, by construction, that node's provenance cannot be needed by any other node. We call this *bottom-up* pruning. It is interesting to ask if *top-down* pruning ever makes sense. On a long-running system, it might be useful to prune provenance to present the illusion that provenance history began at some time T , later than the actual beginning of that system's provenance. Alternately, it might be useful to declare some node as the new eldest ancestor and remove all its parental provenance.

Intermediate files provide another opportunity for pruning. Files that can be easily recreated by capturing the entire process that created them are good candidates for provenance pruning. For example, in a build environment, the object files can be recreated if necessary, so it may not be necessary to keep their provenance.

5.1 Policies

The pruning strategies described in the previous section implement policy decisions. Such policy decisions should be site-specific. Storage policies might place limits on the absolute amount of provenance retained or limit provenance as a proportion of the data it describes. Retention policies dictate when provenance can be erased, e.g., when no file in its ancestry remains.

5.2 Other Provenance Reduction Strategies

Supernodes Short of erasing provenance, it is sometimes possible to compact or summarize existing provenance to save space. For example, it might be desirable to compress the provenance for a subtree whose internal nodes have been deleted. In this case, the system combines the provenance from internal nodes into a *supernode* for the child.

Virtual Nodes. Some collections of attributes might recur frequently. These attributes can be combined and included in a *virtual node* and referenced where applicable. We already use an approach similar to this in our PASS prototype. We store command lines and environments in their own database and refer to them by a unique ID number [18]. An alternate and more general representation would

be to represent them as a provenanced object or virtual node. This approach provides a lossless storage reduction method.

Removing Attributes. Rather than combining attributes, we might chose to remove attributes if we can identify that they are irrelevant. As in pruning, attributes could be removed during collection or later on. Irrelevant attributes could be removed during collection or during later pruning. Identifying attributes that do not need to be recorded in provenance is another site-specific policy decision.

5.3 Pruning in PASS

In our PASS prototype, we do not yet provide a pruning policy specification mechanism. Instead, we provide a utility, *p trunc*, the enables a user to explicitly truncate provenance, eliminating uninteresting ancestry [18].

6 Privacy and Security

Automatic provenance collection presents serious privacy challenges, because such a system collects significant information about its users. Replying to an email message by pasting a paragraph from a web page might cause the system to capture the sender of the original email and the time it was received; the time the message was read; the web addresses the user visited; and any intermediate drafts that were composed but not sent.

Information leakage is the primary privacy risk introduced by automatic collection. By its very nature, automatically collected provenance is *invisible* to users, and this invisibility provides an easy channel through which sensitive information can be released. For example, consider a manager composing an employee review that includes input solicited from the employee's colleagues. Presenting the review and its provenance to the employee reveals the identify of the colleagues who contributed to the review [14]. A manager who had to explicitly disclose the review's provenance to the system would be unlikely to make such a mistake.

Deciding what provenance should be captured, where it should be stored, how that store should be protected, and when (if ever) the stored provenance should be purged are current research areas. For example, provenance is sometimes stored in the document itself as a header [19] or in an alternative data stream [16], making it easy to keep document and provenance together. This approach enables privacy-friendly systems and applications that automatically detect and prevent violations of privacy or data sharing policies [30]. On the other hand, provenance that travels invisibly with a document might also compromise privacy if users are unaware of its existence.

While no provenance-aware storage systems are in widespread use, both the Microsoft Word and Adobe Acrobat file formats store hidden data that reveals elements of a document's history, a kind of provenance. Experience with these formats illustrates that automatic collection introduces significant privacy risks:

- In June 2000, *The New York Times* posted an Acrobat file on its website containing names of Iranians who had assisted the CIA in the 1953 Iranian coup. Although the paper had tried to remove the names from the Acrobat file, the information was recovered and posted in an unredacted form on another website [31].
- In June 2002, the US Justice Department released a “Workplace Diversity” report that contained embarrassing information that the Department had attempted to delete [6].
- In March 2004, the SCO Group distributed Microsoft Word files to journalists containing hidden text that revealed the company’s legal strategy in its anti-Linux lawsuits[27].

On the surface, automated provenance collection also violates many traditional principles of Fair Information Practice [20]. For example, the *collection limitation* principle is violated, because no limits are placed on data collection. The *purpose specification* is violated, because no purpose for the data is specified at the time of collection. On the other hand, FIP principles such as *individual participation* and *security safeguards* could be used as requirements guidelines when designing systems for automatically collecting provenance. Embracing automatic provenance collection necessitates understanding and addressing the privacy implications.

We understood early that provenance security was both crucial and under-researched. We do not provide any access controls in our current prototype, and our early adopters are all users who freely share their data and analyses. In parallel, we undertook a small pilot project to identify the key features a provenance security model requires. That study revealed that we need two independent security models: one that provides conventional access control over provenance attributes and a second that provides access control over the branches of the ancestry tree [3]. We are currently implementing these models and studying the challenges in composing the two models.

7 Related Work

In Sections 2.1 and 2.2 we discussed alternate approaches to provenance. In this section, we discuss a few other approaches and related technologies that influence observed provenance systems or from which we can take advantage of prior knowledge.

7.1 Versioning File Systems

As mentioned earlier, the versioning challenges of observed provenance systems are similar to those in versioning file systems. The Elephant file system creates a new version of a file on every write and does not immediately erase old versions [25]. As such, it highlights many of the issues we encounter with versions. Like PASS, Elephant must cope with the overhead associated with creating a new version on every write, and makes pruning essential. Elephant supports three

pruning policies: keep all, one, or landmarks. Unlike PASS, Elephant versions file (and directory) data, and it does not gather provenance information.

7.2 Observed Provenance

PASS is not the only system to collect low level operations and attempt to infer semantic meaning from them. Both the Lineage File System and Transparent Result Caching take this approach.

Transparent result caching (TREC) captures system calls and tracks process lineage including parent processes, child processes, input files and output files [28]. Unlike PASS, all tracing occurs in user space. Read and write system calls are not intercepted. TREC relies instead on the open mode to infer whether files are inputs or outputs, a tradeoff between performance and accuracy. Both systems support makefile generation and detection of changed dependencies, but PASS provides additional query support unavailable in TREC.

Like PASS, the Lineage File System focuses on executables, command lines and input files as the source of provenance [14]. Unlike PASS, it ignores the hardware and software environment in which such processes run. A second, and perhaps more important, difference is that provenance collection is delayed in the Lineage File System and it is performed by a user-level thread that writes the lineage data to an external database. As a result, the tight coupling we require between data and provenance is lost, as is a significant part of the benefit. Since the Lineage File System stores its lineage records in a relational database, the query language is SQL. In our implementation, we use a simple key/value storage schema so that a variety of schema layers can be provided.

7.3 Exposing Semantic Knowledge

The observed systems considered thus far all try to infer semantic knowledge from a collection of events. Another is to isolate the semantic knowledge as a user specified component. Magpie combines observed provenance recorded in trace logs with a user specified event schema to construct a workflow model [1]. The user specified event schema specifies the rules for combining related events. For example, specifying which filesystem read and write operations correspond to a specific webserver request. Magpie differs from PASS in its use of a user specified event schema and the recording of provenance in separate trace files.

8 Conclusions

Automatic provenance collection is an important and powerful technique that complements existing provenance solutions. However, it carries challenges that do not appear in these other systems. Our group has developed a provenance-aware storage system prototype that is now ready for limited experimental use. We encourage the community to try our prototype, develop an appreciation for the power of automatic collection, and tackle some of the fundamental research challenges that remain. This area holds great promise, but only through the

construction of a variety of systems that automatically collect provenance at different levels, from the operating system to provenance aware applications, will we identify the right solutions to the challenges outlined here.

References

1. P. T. Barham, A. Donnelly, R. Isaacs, and R. Mortier. Using magpie for request extraction and workload modelling. In *OSDI*, pages 259–272, 2004.
2. K.-K. Muniswamy-Reddy, and M. Seltzer. Coping with cycles in provenance. <http://www.eecs.harvard.edu/~syrah/pass/pubs/cycles.pdf>.
3. U. Braun and A. Shinnar. A Security Model for Provenance. Technical Report TR-04-06, Harvard University, Jan. 2006.
4. Brian Cornell and Peter Dinda and Fabi+n Bustamante. Wayback: A User-level Versioning File System for Linux. In *Proceedings of the USENIX 2004 Annual Technical Conference, FREENIX Track*, 2004.
5. ClearCase. <http://www.ibm.org/software/awdtools/clearcase>.
6. R. Edmonds. Justice department hid parts of report criticizing diversity effort. *Associated Press*, Oct. 31 2003.
7. I. Foster, J. Voeckler, M. Wilde, and Y. Zhao. The Virtual Data Grid: A New Model and Architecture for Data-Intensive Collaboration. In *CIDR*, Asilomar, CA, Jan. 2003.
8. J. Frew and R. Bose. Earth system science workbench: A data management infrastructure for earth science products. In *Proceedings of the 13th International Conference on Scientific and Statistical Database Management*, pages 180–189. IEEE Computer Society, 2001.
9. GenePattern. <http://www.broad.mit.edu/cancer/software/genepattern>.
10. A. Heydon, R. Levin, T. Mann, and Y. Yu. The Vesta Approach to Software Configuration Management. Technical Report 168, Compaq Systems Research Center, March 2001.
11. D. Hitz, J. Lau, and M. Malcolm. File System Design for an NFS File Server Appliance. In *Proceedings of the USENIX Winter Technical Conference*, pages 235–245, January 1994.
12. K. Muniswamy-Reddy and C. P. Wright and A. Himmer and E. Zadok. A Versatile and User-Oriented Versioning File System. In *Proceedings of the Third USENIX Conference on File and Storage Technologies (FAST 2004)*, San Francisco, CA, March/April 2004.
13. E. K. Lee and C. A. Thekkath. Petal: Distributed virtual disks. In *Proceedings of the Seventh International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS-7)*, pages 84–92, Cambridge, MA, 1996.
14. Lineage File System. <http://crypto.stanford.edu/~cao/lineage.html>.
15. K. McCoy. *VMS File System Internals*. Digital Press, 1990.
16. Microsoft. How to use ntfs alternate data streams. July 13 2004.
17. S. Muchnick. *Advanced Compiler Design and Implementation*, chapter 8. Morgan Kaufmann, 1997.
18. K.-K. Muniswamy-Reddy, D. A. Holland, U. Braun, and M. Seltzer. Provenance-aware storage systems. In *Proceedings of the 2006 USENIX Annual Technical Conference*, June 2006.
19. Nost. Definition of the flexible image transport system (FITS), 1999.

20. Organisation for Economic Co-operation and Development. Guidelines on the protection of privacy and transborder flows of personal data, 1980.
21. C. Pancerella et al. Metadata in the Collaboratory for Multi-scale Chemical Science. In *Dublin Core Conference*, Seattle, WA, 2003.
22. Z. N. J. Peterson and R. C. Burns. Ext3cow: The design, Implementation, and Analysis of Metadata for a Time-Shifting File System. Technical Report HSSL-2003-03, Computer Science Department, The Johns Hopkins University, 2003. <http://hssl.cs.jhu.edu/papers/peterson-ext3cow03.pdf>.
23. Provenance aware service oriented architecture. <http://twiki.pasoa.ecs.soton.ac.uk/bin/view/PASOA/WebHome>.
24. S. Quinlan and S. Dorward. Venti: a new approach to archival storage. In *Proceedings of First USENIX conference on File and Storage Technologies*, pages 89–101, January 2002.
25. D. J. Santry, M. J. Feeley, N. C. Hutchinson, and A. C. Veitch. Elephant: The file system that never forgets. In *Workshop on Hot Topics in Operating Systems*, pages 2–7, 1999.
26. J. Seward. Valgrind, an open-source memory debugger for GNU/Linux. <http://valgrind.org>, 2005.
27. S. Shankland and S. Ard. Document shows SCO prepped lawsuit against BofA. *News.Com*, Mar. 4 2004.
28. A. Vahdat and T. Anderson. Transparent result caching. Technical Report CSD-97-974, 8, 1997.
29. M. Wan, A. Rajasekar, and W. Schroeder. An Overview of the SRB 3.0: the Federated MCAT. <http://www.npaci.edu/DICE/SRB/FedMcat.html>, September 2003.
30. D. J. Weitzner, H. Abelson, T. Berners-Lee, C. Hanson, J. Hendler, L. Kagal, D. L. McGuinness, G. J. Sussman, and K. K. Waterman. Transparent accountable data mining: New strategies for privacy protection. Technical report, Massachusetts Institute of Technology Computer Science and Artificial Intelligence Laboratory, 2006.
31. E. Wong. Web site lists Iran coup names. *The New York Times*, June 24 2000.
32. J. Zhao, M. Goble, C. and Greenwood, C. Wroe, and R. Stevens. Annotating, linking and browsing provenance logs for e-science.

Electronically Querying for the Provenance of Entities

Simon Miles

School of Electronics and Computer Science
University of Southampton
Southampton, SO17 1BJ, UK
`sm@ecs.soton.ac.uk`

Abstract. The provenance of entities, whether electronic data or physical artefacts, is crucial information in practically all domains, including science, business and art. The increased use of software in automating activities provides the opportunity to add greatly to the amount we can know about an entity's history and the process by which it came to be as it is. However, it also presents difficulties: querying for the provenance of an entity could potentially return detailed information stretching back to the beginning of time, and most of it irrelevant to the querier. In this paper, we define the concept of *provenance query* and describe techniques that allow us to perform *scoped provenance queries*.

1 Introduction

In order to understand, apply, or judge the accuracy or authenticity of an entity, whether electronic or physical, it is often crucial to know its *provenance*, i.e. from where it originated, how it was produced and what has happened to it since creation [9]. For example, in e-Science, to determine if an experiment's results are accurate, we look at the rigour of the experiment's process.

However, the amount of information making up the provenance of an entity may be vast [4]. The details of *everything* that ultimately caused an entity to be as it is would, generally, be an unmanageable amount. For example, to give the full provenance of an experiment's results, we have to describe the process that produced the materials used in the experiment, the provenance of materials used in producing *those* materials, the devices and software used in the experiment and their settings, the origin of the experiment design etc. Ultimately, if enough information was available, we would include details of processes back to the beginning of time. Similarly, given enough information, we could give finer and finer grained information on the processes that led to an entity, e.g. not just documenting that a sample was tested to see if a chemical was present, but the procedure by which this is done, the molecular interactions that took place in the testing procedure and so on. All the information about the provenance of an entity is potentially useful for someone with a particular question about that entity, but providing it all in all cases would be counter-productive.

Instead, anyone requiring the provenance of an entity should be able to get it by expressing the request as a *query* and *scoping* that query so that only the information relevant to them is returned. The contributions of this paper are to specify what a provenance query consists of and how it can be expressed, an algorithm for processing a provenance query to obtain the results and the form taken by results of a provenance query.

In the next section, we look at earlier work on retrieving provenance. Section 3 examines the properties data must have in order to extract the provenance of an entity from it, and the delineations used to scope a provenance query. Section 4 specifies a model of a provenance query and its results, illustrated by example in Section 5. We compare our work with others in Section 6 and conclude in Section 7.

2 Provenance

From existing work on provenance, we can derive two general properties. First, *the provenance of an entity depends on its relationships to other entities*. For instance, the results of an experiment are produced from processing intermediate data in an experiment, which are in turn produced from the inputs to the experiment. In myGrid [12], an entity is marked as being *derived from* another entity which, in turn, is derived from other entities. Using this approach, the provenance of an entity is determined from relations of that entity to other entities it is directly or indirectly derived from.

Second, *the provenance of an entity can vary in scope and, at widest scope, is everything that has had a causal effect on the entity*. Buneman [5] describes two differently scoped types of provenance, named *where* and *why provenance*, applied to database query results. Applied within the closed world of a database system, *where provenance* is the origin of the entity and its components, while *why provenance* is all data having a causal effect on the entity.

We argue that the provenance of an entity can be seen as a *process* that leads to the entity being in its current state. Full information on that process would include the origin of and anything having an effect on the entity. To obtain the provenance, we need documentation of how that process occurred, at some level of abstraction, e.g. we may infer the process from the database query or workflow whose execution resulted in the entity, or we may have details of each action performed by actors in the process. Documentation of past processes is called *process documentation*.

In our model, we separate two concepts: process documentation, documentation of executed processes, and provenance, a description of the process leading to an entity which is determined from process documentation. The separation allows us to tailor systems to best meet the requirements on each. For example, the details of a process may not be recoverable after it has completed, so should be documented as completely and accurately as possible during or immediately after [7], while provenance can be determined as long as process documentation is preserved unmodified, so should be scoped to include only that relevant to the querier.

3 Process Documentation

A provenance query is executed over process documentation and, in order to implement a scoped query, documentation must be structured so that it can be determined whether any piece of information is part of the provenance of an entity. Below, we introduce the properties and delineations of process documentation that can be used to answer provenance queries.

3.1 Assertions and Temporality

An *actor* is anything that performs actions, and each piece of process documentation is created by an actor: by recording data on processes it has taken part in or inferring what happened from information from other actors. We can provide no guarantee that documentation accurately reflects what occurred, so documentation is actually *assertions* by actors. A *p-assertion* is an assertion about a process and process documentation is comprised of *p-assertions*.

A process includes multiple *events* and the provenance of an entity as it exists as one event occurs is different from the provenance of the same entity later or earlier. For example, the provenance of a hospital patient, the process which led to that patient being in a particular state [1], would be different for the patient up to an operation starting and up to the operation finishing, because after the latter event the provenance includes data about the operation. An event that an assertion provides documentation about need not be instantaneous, but entities documented in the assertion must not change over the course of the event (else the provenance would be ambiguous). In our approach, the provenance of an entity is always the provenance of an entity as it exists when an event occurs, called the *entity's provenance up to the event*. In order to determine if a p-assertion is part of an entity's provenance up to a given event, it must be clear which event it documents.

3.2 Relationships

As stated in Section 2, the provenance of an entity depends on its relationships to other entities and so process documentation must include these relationships to be used in determining provenance. Relationships are directional, so we identify one entity in the relationship as the *subject* and the others as *objects*, e.g. the subject “12” *was the result of summing* objects “5” and “7”. As one p-assertion may document an event in which several entities were involved, a relationship is between *parts* of p-assertions. A *p-assertion data item* is an entity within a p-assertion and a relationship is between two or more p-assertion data items. A p-assertion documenting the relationship of an entity in a p-assertion to entities in other p-assertions is a *relationship p-assertion*.

Relationship Types. Relationships can be of different types, the most abstract being a *causal* relationship, i.e. *E* was caused by *C*. While causal relationships are all that is needed to determine which documentation is part of an entity's provenance, they are inadequate for scoping the query. We need more information on *how* one entity is related to another to determine if some process

documentation is relevant. Therefore, *functional relationships* between entities can be asserted, stating that an entity was produced by performing a function on other entities, e.g. an actor may assert that a value produced in an experiment is a product of measuring the weight of a sample (there is a functional relationship between the value and the sample).

Parameter Names. A p-assertion often documents a relationship in the world in which the entities being related play *roles* in that relationship, e.g. in asserting that the results of a *divide* operation were derived from its inputs, we must mark the entities involved with the roles they play: divisor and dividend for the inputs, quotient and remainder for the outputs. The name of an entity's role in a relationship is a *parameter name*, which must be asserted with relationships for the documentation to be interpretable.

4 Provenance Queries

Below, we specify a model for expressing provenance queries. To execute a query, a *provenance query request* is sent to a *provenance query engine* by a *querying actor*. A provenance query request includes a *query data handle*, identifying the entity of which to find the provenance, and a *relationship target filter*, specifying the query's scope. These are shown in Table 1, and described in full below.

4.1 Query Data Handles

When a querying actor asks for an entity's provenance, it identifies the entity such that a query engine can find documentation of the entity. The identification is called a *query data handle*. For the actor, a query data handle identifies an entity at a given event. For the engine, a query data handle identifies a search for p-assertion data items in process documentation. A handle comprises three parts, discussed below.

A p-assertion documents an event in a manner dependent on the way the system is modelled. Part of a handle is an *event search*, a search for documentation of the event in which the entity occurred. Within documentation of that event, an *entity search* finds p-assertion data items documenting the entity. The *search space* of a data handle identifies the set of process documentation to be searched.

4.2 Relationship Target Filters

A *relationship target filter* is used to scope a query to the part of a process of interest to the querying actor. More concretely, we can say that, given that a p-assertion data item has been identified as part of a query's results, and that that data item is related to other data items (by relationship p-assertions), the relationship target filter specifies which related data items should also be included in the results.

A *relationship target* is a set of properties of a data item that is the object of a relationship p-assertion. The properties, e.g. the event that the p-assertion

Table 1. Data comprising a provenance query request

Provenance Query Request	
Query Data Handle	A search over process documentation to find the record of an entity at a given event for which the querying actor wishes to find the provenance.
Relationship Target Filter	Criteria by which the querying actor specifies whether any given entity in the documentation, and its relations, should be included in the query results.
Starting Search Space	The process documentation set from which to start searching for the provenance of the entity.
Query Data Handle	
Event Search	A search for the relevant event involving the entity.
Entity Search	A search in p-assertions for data items documenting the entity.
Search Space	The process documentation set that will be searched over.
Relationship Target	
Event	The event which the p-assertion is documenting.
Global P-Assertion Key	The globally unique identifier for the p-assertion.
Parameter Name	The role played by the object in the relationship.
Provenance Store Address	The address where the p-assertion is stored.
Data Accessor	The location of the data item within the p-assertion.
Relationship	The relation (name) of which this target is an object.
Asserter	The asserting actor's identity, if known.
P-Assertion Content	The content of the p-assertion (the actual documentation).

documents or the asserter's identity, are those described in Section 3 and are listed in Table 1. A relationship target filter is a function over a *relationship target* returning a boolean value specifying whether the relationship target is within scope. For example, a relationship target filter may return false for relationship targets where the asserter is a particular, untrusted, source. In this case, the provenance query results exclude all p-assertions by that asserter and p-assertions iteratively related to those assertions.

4.3 Provenance Query Results

A provenance query request is processed as follows. First, perform the search expressed by the query data handle to find a set of p-assertion data items. For each relationship of which one of those items is a subject, execute the relationship target filter on the information about each object of the relationship (i.e. each relationship target). Where an object is accepted by the relationship target filter, recursively apply the filter to objects of its own relationships. The final results of the query are comprised of two parts: the p-assertion data items from the query data handle search (the *start* data items); and, for every relationship object accepted by the filter, the relationship between that object and the subject in that relationship.

A query engine, with instantiations of the above model in XML, has been implemented as part of an open source distribution. Due to space restrictions,

we do not describe it here, but refer readers to www.pasoa.org, from which it can be downloaded, and a link to a functional specification of the engine's interface can be found.

5 Case Study

To demonstrate provenance queries completely but concisely, we use a simple, contrived example. A workflow, shown in Figure 1, is run and process documentation generated: a GUI actor calls an Averager service with two values (7, 5); Averager sums them and calls Divider with 12 as divisor, 2 as dividend; Divider sends the answer 6 to Averager, which sends it to the GUI; the GUI sends 6 to Store, along with the file location, e.g. `file1`, at which to store it.

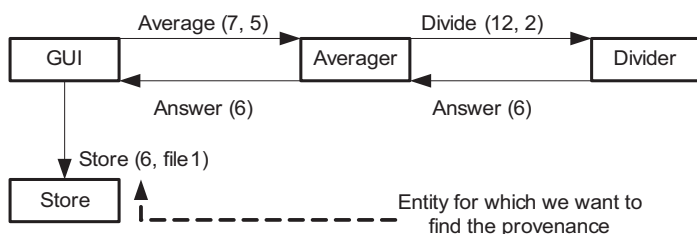


Fig. 1. An example workflow

Two types of p-assertion are recorded by actors in the scenario. An *interaction p-assertion* is a copy of the message sent between two actors. A relationship p-assertion asserts a relationship between data items exchanged in messages (and so documented in interaction p-assertions). A querying actor can derive a causal relationship from an interaction p-assertion: a message being received is caused by the same message being sent. The events to which p-assertions are declared to apply are the sending or receiving of messages. Each actor in the workflow makes interaction p-assertions about every message it sends and receives.

For this scenario, we show the results of a provenance query scoped in different ways. The query data handle represents a search for the “data requested to be stored at location `file1` (*entity search*) at the event of the Store actor receiving a message (*event search*).” The query returns results in which the start p-assertion data item is the 6 in the message from GUI to Store, and, if unrestricted in scope, includes all relationships recorded. The relationships form a directed acyclic graph linking data items in exchanged messages, shown in Figure 2, with the start data item shown at the bottom. Broken lines between data items depict asserted relationships, labelled with their functional relationships. Parameter names, “divisor” and “dividend” are shown for the objects of the “Division of” relation.

One way to scope the provenance is to exclude a type of relationship, e.g. by excluding “Average of”, that relationship is removed from the graph returned by

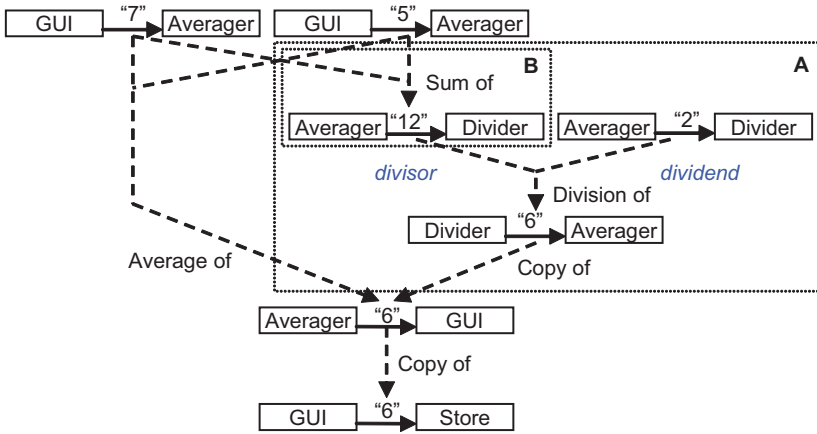


Fig. 2. The provenance of the data stored in `file1`

the query. As a comparable practical example, in asking for the provenance of a journal article, we may want to include the origins of its content and not the origins of the paper on which it is printed: these are types of relation from the journal paper to other entities. Another scope excludes the internal operations of an actor, such as `Averager`. This removes the documentation shown in box A in Figure 2 from the results. Comparably, we may want to know from which database we downloaded data, but not the database query: we make the results more coarse-grained by hiding part of the process. Finally, we can scope on the role that data plays in the process by excluding all “divisors” from the results (as specified in the parameter name), so removing the relationships shown in box B. Comparably, in querying for the provenance of compressed data, we may wish to know the data before compression but not the compression algorithm used: two roles in the compression function. The true benefits of the scoping process require a more detailed example than is possible to give in this paper, and will be in future work.

6 Related Work

In related work, provenance queries of a form are executed, but the approaches differ considerably from ours. In most systems, the mechanism is assumed to be an unspecified query of a database containing documentation [2,11], but a few specify another mechanism. Some systems, e.g. lab information management systems (LIMS) and work on deriving provenance data from database queries [5], operate in closed systems, allowing extra positive assumptions to be made on the quality of the query results, but preventing determination of an entity’s provenance where the process that led to it spans multiple systems. Also, results cannot be scoped to arbitrary levels. Similarly, approaches for which queries are made with regards to workflows rather than their results [3], are limited to answering questions of the provenance of data within a workflow.

myGrid [12] and CombeChem [8] have advocated storing process documentation as functional and derivation relations between entities, named by URIs, as RDF triples. This has the advantage that query languages, e.g. SPARQL [10], exist to query relationship-based data, but there is no distinction between events in an entity's lifetime, e.g. in asking for the provenance of an experiment result, is an actor asking for how the result was produced, how it has been used as input to other processes (such as publication) since, or both? This means that the data on which a query can be scoped is limited. Some approaches visualise the provenance of data or records of workflow execution, allowing users to iteratively hide or retrieve data on the basis of what they have previously seen [6]. While this is productive and simple, it is only practical where the user is the direct consumer of the data, and does not address the problem where documentation is to be further processed to answer complex questions.

7 Conclusions

The provenance of an entity is essential, in many domains, for understanding and evaluating that entity, but the amount of information that makes up the provenance of an entity could be huge, so the results need to be *scoped* to retrieve only what is relevant for the querier. To execute a scoped provenance query, documentation over which the query is executed must have particular characteristics. In this paper, we have specified how a scoped provenance query can be expressed and executed. We exposed the necessary characteristics of the documentation over which the query is processed, criteria available for scoping, a data model for query requests and the algorithm used to execute a query.

This research is funded by the PASOA project (EPSRC GR/S67623/01).

References

1. Sergio Alvarez, Javier Vazquez-Salceda, Tamas Kifor, Laszlo Z. Varga, and Steven Willmott. Applying provenance in distributed organ transplant management. In *this volume*, 2006.
2. Rajendra Bose and James Frew. Lineage retrieval for scientific data processing: A survey. *ACM Computing Surveys*, 37(1):1–28, March 2005.
3. Shawn Bowers, Timothy McPhillips, Bertram Ludascher, Shirley Cohen, and Susan B. Davidson. A model for user-oriented data provenance in pipelined scientific workflows. In *this volume*, 2006.
4. Miguel Branco and Luc Moreau. Enabling provenance on large scale e-science applications. In *this volume*, 2006.
5. P. Buneman, S. Khanna, and W.C. Tan. Why and where: A characterization of data provenance. In *Int. Conf. on Databases Theory (ICDT)*, 2001.
6. Juliana Freire, Claudio T. Silva, Steven P. Callahan, Emanuele Santos, Carlos E. Scheidegger, and Huy T. Vo. Managing rapidly-evolving scientific workflows. In *this volume*, 2006.

7. Paul Groth, Simon Miles, and Steve Munroe. Principles of high quality documentation for provenance: A philosophical discussion. In *this volume*, 2006.
8. Gareth Hughes, Hugo Mills, David de Roure, Jeremy G. Frey, Luc Moreau, m.c. schraefel, Graham Smith, and Ed Zaluska. The semantic smart laboratory: a system for supporting the chemical scientist. *Organic and Biomolecular Chemistry*, 2(2):1–10, 2004.
9. Simon Miles, Paul Groth, Miguel Branco, and Luc Moreau. The requirements of recording and using provenance in e-science experiments. Technical report, University of Southampton, 2005.
10. Eric Prud'hommeaux and Andy Seaborne. Sparql query language for rdf. <http://www.w3.org/TR/rdf-sparql-query/>, 2006.
11. Y. Simmhan, B. Plale, and D. Gannon. A survey of data provenance in e-science. *SIGMOD Record*, 34(3):31–36, 2005.
12. J. Zhao, C. Goble, M. Greenwood, C. Wroe, and R. Stevens. Annotating, linking and browsing provenance logs for e-science. In *Proc. of the Workshop on Semantic Web Technologies for Searching and Retrieving Scientific Data*, October 2003.

AstroDAS: Sharing Assertions Across Astronomy Catalogues Through Distributed Annotation

Rajendra Bose¹, Robert G. Mann², and Diego Prina-Ricotti³

¹ U.K. Digital Curation Centre and School of Informatics, University of Edinburgh
rbose@inf.ed.ac.uk

² Institute for Astronomy, University of Edinburgh
rgm@roe.ac.uk

³ Dipartimento di Informatica e Automazione, Università di Roma Tre
dpricott@inf.ed.ac.uk

Abstract. As diverse scientific data collections migrate online, researchers want the ability to share their assertions regarding the entities that span these disparate databases. We focus on a case study provided by the astronomical community’s Virtual Observatory effort to investigate the use of annotation to record and share the celestial object mappings asserted by different research groups. The prototype for our Astronomy Distributed Annotation System (AstroDAS) complements the existing OpenSkyQuery tools for federated database queries, and provides web service methods to allow clients to create and store mapping annotations as relational database tuples on annotation servers. We expect the mechanisms for creating and querying annotations in AstroDAS can be extended to assist with tasks other than entity mapping, in other domains with relational data sources.

1 Introduction

Research activity in a number of scientific domains includes organizing and analyzing content culled from diverse collections of online data. Usually, portions of different databases are extracted and placed into personal or research computing environments within a particular organization or group (which may include autonomous relational or XML databases, spreadsheets, and so forth) where the analysis is conducted. We focus on a case study from astronomy, where a group analyzes data from different source databases, and then *asserts* mappings between entities in the data sources. Others who access the source databases may benefit from receiving these assertions as feedback.

Central to the astronomical community’s concept of a global “Virtual Observatory” (VO) is the ability to identify records in databases of astronomical observations as referring to the same celestial object: for example, a typical VO use case might start by matching entries from an optical, an X-ray and a radio database, so that an astronomer could analyze the multi-wavelength properties of a particular class of galaxy. The nascent VO already includes web services which implement the matching of catalogues by spatial proximity alone, but the general catalogue matching problem is

more difficult—and more computationally expensive—than that. This problem motivates our investigation of recording, through *annotations*, the assertions made regarding the associations between entries in different databases, such that they can be re-used by third parties.

In our work, we build on the ideas introduced by the Distributed sequence Annotation System (DAS, later BioDAS) [1]. BioDAS is specifically designed to facilitate genome sequence annotation, however, so while the concepts it introduces are valuable, the approach and means of implementation for this system are not suitable for sharing assertions in other scientific domains. Thus we first introduce a general framework to help clarify and compare other types of annotation systems. Following this, we describe our design of a system for distributed annotation in the context of OpenSkyQuery, a prototype implementation of the International Virtual Observatory Alliance (IVOA) SkyNode specification for nodes in the federated VO. We discuss the topic of matching celestial objects in astronomy catalogues, and describe how the prototype for our Astronomy Distributed Annotation System (AstroDAS) complements existing OpenSkyQuery tools for federated database queries. We close with a brief discussion of related work.

2 A Framework for Annotation

The BioDAS protocol for genome sequence annotations is not directly applicable to the process of annotation in other areas of science. In order to contrast other types of annotation systems we suggest an informal framework that consists of the following basic components:

An *annotation* is some set of data elements that is added to an existing *base* or *target* that possesses structure; the base or target structure can be described by some reference system; and the *point* or *location of attachment* of an annotation can be described using this same reference system.

These various components of BioDAS are summarized in column (2) of Table 1: the annotation target is a (conceptualized or idealized) genome, the target structure consists of a linear sequence of nucleotides (base pairs), and the reference system for the target is a linear coordinate system of base pair numbers. Annotations are usually the identification of particular genes or their products (such as proteins), and the location of annotation attachment is designated by start and stop nucleotide positions (base pair numbers). In this context, the purpose of annotations is to collect and interpret the results of numerous biological experiments or computer algorithms.

Consider the very different example for the annotation of medical images described in [2] (Table 1, column (3)). Here the annotation target is a Human Brain Project (HBP) image, the target structure is a 2D array of pixels, the reference system for the target is a 2D coordinate system of X,Y pixel values, annotations are concepts (for example, controlled medical vocabulary terms), and the location of attachment is a 2D region of interest described by reference to the coordinate system. Other systems with a similar purpose exist, including the Edinburgh Mouse Atlas [3].

Table 1. Annotation framework and system comparison

(1) Framework components	(2) BioDAS	(3) HBP image annotation	(4) AstroDAS
Annotation target:			
what	genome	brain image	celestial object in astronomy catalogue (RDBMS tuple)
structure	sequence of base pairs	pixel array	catalogue (RDBMS) schema
reference system	linear coordinate system of base pair numbers	2D coordinate system of pixel X,Y values	catalogue+celestial object id (tuple key)
Annotation:			
what	gene or gene product	domain-specific concept	mapping to a celestial object in a different catalogue
location of attachment	start, stop base pair numbers	2D shape	specific catalogue +celestial object id
purpose	collect and interpret research results on genome	link Web-accessible images to, and query on, concepts	share assertions of celestial object matches across different astronomy catalogues

Column (4) of Table 1 summarizes the components of the AstroDAS annotation system prototype described in the remainder of this paper; these components are somewhat similar to the relational annotation work discussed in [4, 5]. In this case, the annotation target is an entry (tuple) in an astronomy catalogue implemented with a relational database management system (RDBMS), the target structure is given by the catalogue RDBMS schema, and the reference system for the target is provided by catalogue and database object ids (tuple keys), possibly combined with the name or location of specific attributes. Annotations are other tuples that map one catalogue object to another catalogue object, and the location of attachment is a specific catalogue and object id combination. In our work we use relational database framework components because we focus on providing annotation over the federated relational databases in the OpenSkyQuery system, described in the following sections.

3 Matching Celestial Objects in Astronomy Catalogues

Over the past several decades, collections or *catalogues* of celestial object observations, recorded by disparate telescopes and other instruments over various time periods, have migrated online. Astronomy catalogues have different schemas but are usually organized according to a star schema consisting of a primary relation of celestial objects that serves as the basis for most joins. As expected, each tuple in the

primary relation contains a key or *celestial object identifier* (id) that is unique within a specific catalogue. Most astronomy catalogues are updated through large data releases, for example every three to six months, but we assume here that all ids within catalogues are persistent and stable.

Newer astronomy instruments are designed to provide *sky surveys* of large portions of the celestial sphere. While older catalogues consist of hundreds or thousands of objects, these new instruments generate on the order of hundreds of gigabytes of data per day, contributing to catalogues (also known as survey archives) that record the observations of hundreds of millions of celestial objects. The recorded location of a celestial object may vary slightly from catalogue to catalogue due to unavoidable measurement error at the instrument level [6].

The OpenSkyQuery system aims to federate astronomy catalogues and archives through the use of web services and a wrapper-mediator architecture [7]. Federated queries are executed by a portal application that communicates to registered *sky nodes* (wrappers for astronomy catalogues) through a standard web service interface (See lower right of Figure 1. The three sky nodes shown correspond to the catalogues for the Sloan Digital Sky Survey (SDSS), the Two Micron All Sky Survey (TWO MASS), and the U.S. Naval Observatory USNO-B1.0 catalogue (USNOB).)

The queries are expressed in ADQL (Astronomical Data Query Language) [8], which is based on a subset of SQL92 with two extra keywords: *Region* and *XMatch*. The *Region* keyword allows the user to specify the spatial extent of the search using celestial coordinates. The *XMatch* keyword lets the user specify a measure of probabilistic uncertainty *sigma*; for a given sigma, an X-match query result consists of two or more columns of potentially coincident celestial object ids (and possibly other attributes) [6]. The locations of the resulting celestial object matches essentially cross catalogues (that is, exist across two or more catalogues) within a specified sigma, and therefore match. In this case, however, any further assessment or analysis of X-match results will probably occur in a personal or research computing environment separate from the federated OpenSkyQuery system.

The simple spatial proximity match performed in X-match is inadequate in the general case, where differences in the angular resolution of the data in the two catalogues may mean that a large number of objects from one catalogue (A) will lie within the positional error ellipse of each source in the other (B). In that case, spatial proximity alone cannot judge between the potential counterparts from A of each source in B, and it is necessary to either introduce prior astrophysical knowledge to help identify the most likely match, or use a machine learning algorithm [9, 10]. Machine learning algorithms deduce relationships between the properties of the sources in the two catalogues and can aid the finding of associations between them.

Adding sophistication to the matching algorithm also adds computational cost, and one of the main motivations behind investigating the AstroDAS system is the desirability of having a means of recording, and making available for re-use, associations made by algorithms more complex than the simple X-match function implemented in OpenSkyQuery. This has been studied by Taylor [11], who has implemented as web services a number of cross-matching algorithms which make use of non-spatial attributes as well as simple proximity matching. The design of AstroDAS was influenced by the desire to store the results of these services.

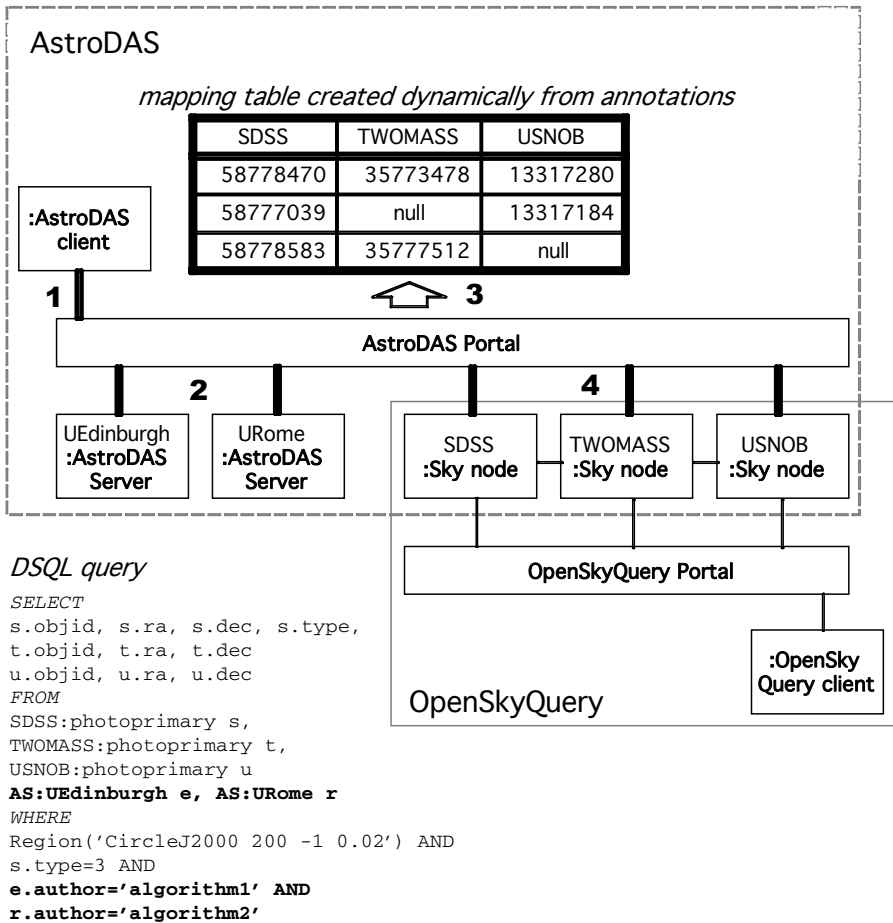


Fig. 1. OpenSkyQuery and AstroDAS architecture

Thus, to provide astronomers with the ability to share their assertions about matching celestial objects directly with their colleagues, we investigate prototypes for AstroDAS, a distributed annotation system that supports queries on annotation, and that is compatible with the federated architecture of OpenSkyQuery.

4 AstroDAS: Asserting Celestial Object Matches Using Distributed Annotation

AstroDAS considers the identification of entities across sky nodes to be a database integration problem. Its solution, inspired in part by BioDAS, features an annotation database with a web service interface to store and query annotations. The use of web services provides interoperability with the rest of the Virtual Observatory. Our

prototype system resolves queries on astronomy catalogues using mapping tables that are dynamically constructed from annotations of celestial object matches.

4.1 Celestial Object Matching as Database Integration

Storing assertions of celestial object matches is essentially the problem of *entity identification* that occurs in traditional database integration scenarios. Entity identification is the ability to find “object instances from different databases which correspond to the same real-world entity” [12]. In database integration, *mapping tables* are commonly employed to store the keys of corresponding tuples that reside in different databases [13].

We make use of mapping tables to express celestial object matches across federated astronomy catalogues. Although the schemas of sky node catalogues differ, we mention in Section 2 that each catalogue contains one primary relation with unique celestial object ids. We thus consider a specific celestial object instance, which we call a *database object*, as the ordered pair: <sky node name, celestial object id>. In our case, mapping tables consist of tuples of database objects that map to one another.

Given the collection of decentralized, read-only sky nodes, and the lack of a central authority to resolve disputed mappings, we use the concept of distributed annotation to provide autonomous research groups with the means to store their own celestial object matches locally. In our approach, a group stores *mapping annotations* that refer to two or more sky node entries (database objects) on their own annotation server. A group’s assertions of celestial object matches can then be shared by the wider community if the group provides online access to their annotation server web service interface. Once this is done, queries on specific sky nodes can include parameters for annotation.

4.2 Storing and Querying Mapping Annotations in AstroDAS

Storing and querying mapping annotations in AstroDAS are achieved through the mechanism of web services. Web services are compatible with the OpenSkyQuery infrastructure adopted by the IVOA; they also facilitate client implementation and promote system interoperability. Similar to OpenSkyQuery, AstroDAS follows the wrapper/mediator architecture whereby a client connects to a portal (mediator) that accesses data on both annotation servers and sky nodes through wrappers implemented with a web service interface (Figure 1). The wrappers provide a common data model and query language, and hide the potential heterogeneity of the different data sources.

Note from Figure 1 that AstroDAS is separate from OpenSkyQuery. The loose coupling of web services means that it has been possible to make AstroDAS interoperable with the OpenSkyQuery system without modifying OpenSkyQuery’s existing code or requiring any action by its developers and maintainers. AstroDAS web service methods exist to allow clients to create and store mapping annotations as relational database tuples on AstroDAS annotation servers (Figure 1, label number 2).

AstroDAS includes the custom query language DSQL (Distributed SQL), similar in purpose to MySQL [14] but more limited in scope. DSQL was designed to retain a syntax as close as possible to ADQL, but *from* clauses in DSQL can specify a list of annotation servers to access annotations from, and *where* clauses in DSQL can specify constraints on mapping annotations.

A DSQL query example is shown in the lower left of Figure 1. The structure is identical to an ADQL query, with the addition of constraints unique to DSQL shown in bold. The hypothetical query shown requests the value of attributes for celestial objects that match according to either the results of “algorithm1” stored on the University of Edinburgh AstroDAS annotation server, or the results of “algorithm2” stored on the University of Rome AstroDAS annotation server.

Methods to retrieve mapping annotations in the form of a mapping table are available. One aspect of the AstroDAS portal design concerns the use of an *inference algorithm* to simplify mapping tables by inferring new mappings from existing ones. Because mappings are transitive, the simplified mapping table to the right of Figure 2 can be inferred from the mapping table to the left, for example.

The execution of a DSQL query similar to the example shown in the lower left of Figure 1 proceeds as follows: A client sends a web service request to the AstroDAS portal mediator with a DSQL query as a parameter (Figure 1, label number 1). The portal parses the DSQL query, generates a query for each annotation server and executes them (Figure 1, label 2). The portal then receives the mapping table results of the annotation queries, and dynamically combines the separate mapping tables into one if necessary (Figure 1, label 3). The mapping table shown in Figure 1, label 3 could have resulted from, for example, the UEdinburgh server storing the annotations:

```
<SDSS, 58778470> maps to <TWOMASS, 35773478>
<SDSS, 58778470> maps to <USNOB, 13317280>
<SDSS, 58777039> maps to <USNOB, 13317184>
```

and the URome server storing the annotation:

```
<SDSS, 58778583> maps to <TWOMASS, 35777512>
```

The inference algorithm is executed on the resulting mapping table if a specific keyword is included in the DSQL query. The mapping table is used to generate the queries for the astronomy catalogues in order to retrieve the actual data. The portal performs the queries by contacting the sky node web service wrappers (Figure 1, label 4). With the query result relations retrieved from the astronomy catalogues, the portal uses the mapping table to combine mapped entities; that is, the data corresponding to the same celestial object is concatenated in the same row of the resulting table. Finally, the portal returns the result to the client (Figure 1, label 1).

Thus, through annotation retrieval, DSQL queries similar to the given example provide a means to perform *entity joins* [15] across different sky nodes. Successive

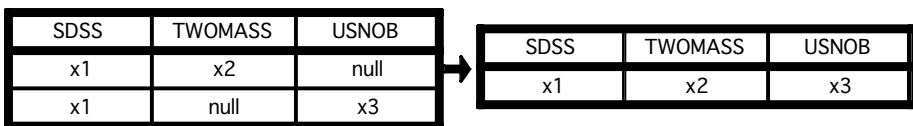


Fig. 2. Applying inference to a mapping table

prototypes of AstroDAS have been implemented by the authors and tested by astronomers from the University of Edinburgh associated with AstroGrid, the UK's VO development project (www.astrogrid.org). The AstroDAS portal and annotation server web service interfaces are implemented through Apache Axis2; the annotation server uses a PostgreSQL or IBM DB2 database, accessed by the web service interface through JDBC.

4.3 AstroDAS v2: Towards Peer-to-Peer Annotation

AstroDAS version 2 is still in development; this successor to AstroDAS is an evolution of the initial architecture to explore the replacement of annotation servers (see Figure 1) with *annotation peers*—nodes in a peer-to-peer (P2P) network of databases. Each peer shares its own set of annotations and cooperates with other nodes for retrieving query results. A distributed annotation system based on a P2P structure is expected to be more scalable than other architectures because the computational load is spread across the peers; the issue of scalability becomes important as the number of research groups sharing their annotation increases.

We refer to the same example DSQL query shown in the lower left of Figure 1, but describe its execution with a network of annotation peers rather than annotation servers. As before, the client sends a web service request to the AstroDAS portal mediator with a DSQL query as a parameter (Figure 1, label number 1). The portal parses the DSQL query, and generates a *single* query for the P2P annotation network that now takes the place of the two annotation servers shown by Figure 1, label 2. The portal then sends a web service request to one annotation node of its choice with the single query as a parameter. This annotation node is called the *coordinator* and is responsible for interacting with the other nodes in order to answer the query. The coordinator returns the mapping table result to the portal, and the DSQL query execution continues as previously described.

The AstroDAS portal logically views the P2P annotation network as a single global mapping table, which integrates all the local mapping tables in the network. Thus the portal only issues a single query on this global mapping table to the coordinator. A typical query could ask for mapping annotations that include: celestial objects belonging to a particular region of the sky; data held on an arbitrary number of skynodes; one or more specific authors; and some minimal degree of reliability. The coordinator determines the annotation peers that are involved with the query and sends an asynchronous web service request to those nodes to start the computation of their local mapping table. The coordinator then determines an execution plan in order to reduce network traffic and starts the execution that contacts the nodes in sequence to integrate the global mapping table from the local ones.

Furthermore, AstroDAS v2 provides a distributed inference algorithm, related to the one described in [13], that computes a global inferred mapping table. The computational load of this algorithm is distributed across all the nodes involved with the query, rather than executing only on the portal (as in the previous AstroDAS version).

5 Related Work

The annotation management system for relational databases presented in [4] and affiliated papers assume that source databases are already populated with annotations, such as location tags for all attributes in every tuple. Although annotations are propagated through the system, direct queries on annotation values are not possible. The Mondrian prototype [5] introduces a system for creating annotations for subsets of tuple attributes, as well as the associations between multiple values; this project includes the design of mechanisms to "query values and annotations alike (in isolation or in unison)." Image annotation projects [2, 3] use relational databases and also offer the ability to make these same types of queries. Annotations for the entity mapping in our work are different from [2, 3, 5]: we create annotations that are both external to their targets of relational tuples and distributed among different groups. Like these projects, however, we share the goal of providing the ability to query on a mixture of data and annotation values.

6 Conclusion

The ultimate aim of AstroDAS is similar to the goal of the earlier BioDAS: to record and share scientific assertions with a wider community. Whereas biologists use annotation to interpret the reference map of a genome, however, astronomers seek to share the mapping of entities derived from their research across established scientific databases. Specifically, astronomers want to be able to share their identification of matching celestial objects within the existing federation of disparate catalogues.

The contribution of our work is to demonstrate how distributed annotation can be used beyond the domain of bioinformatics to assert entity mappings across databases. Successive AstroDAS prototypes suggest the use of annotation servers that complement the existing OpenSkyQuery data access system. They also demonstrate how dynamic mapping tables constructed from stored annotations can serve as the means to map entities across disparate databases. We expect that continuing work will show that AstroDAS mechanisms for creating and querying annotations can be extended to assist with tasks other than entity mapping, in other domains with relational data sources.

Acknowledgements

We would like to thank: John Taylor, Emma Taylor, Martin Hill, and other members of the Wide Field Astronomy Unit, University of Edinburgh; Anastasios Kementsietidis, Floris Geerts and other members of the Database Group in the School of Informatics, University of Edinburgh; and Professor Paolo Atzeni at the Dipartimento di Informatica e Automazione, Università di Roma Tre for their support and help in this work. This project has been supported in part by the UK Digital Curation Centre, which is funded by JISC and the eScience core programme.

References

1. L. D. Stein, S. Eddy, and R. Dowell, "Distributed Sequence Annotation System (DAS) Specification Version 1.53," 21 March 2002. <<http://www.biodas.org/documents/spec.html>>
2. M. Gertz, K.-U. Sattler, F. Gorin, M. Hogarth, and J. Stone, "Annotating Scientific Images: A Concept-based Approach," in *Proceedings of the 14th International Conference on Scientific and Statistical Database Management (SSDBM 2002)*, Edinburgh, Scotland, 2002, pp. 59-68.
3. R. A. Baldock, C. Dubreuil, W. Hill, and D. Davidson, "The Edinburgh Mouse Atlas: Basic Structure and Informatics," in *Bioinformatics: Databases and Systems*, S. I. Letovsky, Ed.: Kluwer Academic Publishers, 1999, pp. 129-140.
4. D. Bhagwat, L. Chiticariu, W. C. Tan, and G. Vijayvargiya, "An Annotation Management System for Relational Databases," in *Proceedings of the VLDB*, Toronto, Canada, 2004, pp. 900-911.
5. F. Geerts, A. Kementsietsidis, and D. Milano, "MONDRIAN: Annotating and querying databases through colors and blocks," in *Proceedings of the ICDE*, 2006.
6. T. Malik, A. S. Szalay, T. Budavari, and A. Thakar, "SkyQuery: A Web Service Approach to Federate Databases," in *Proceedings of the Conference on Innovative Data Systems Research (CIDR)*, Asilomar, CA, 2003, pp. 17-26.
7. T. Budavari, A. S. Szalay, J. Gray, W. O'Mullane, R. Williams, A. Thakar, T. Malik, N. Yasuda, and R. Mann, "Open SkyQuery -- VO Compliant Dynamic Federation of Astronomical Archives," in *Astronomical Data Analysis Software and Systems (ADASS) XIII (ASP Conference Series)*, vol. 314, F. Ochsenbein, M. G. Allen, and D. Egret, Eds. San Francisco: Astronomical Society of the Pacific, 2004, pp. 177-180.
8. IVOA VOQL Working Group, "IVOA Astronomical Data Query Language Version 0.91," IVOA Working Draft 2005-02-25, IVOA, 2004-08-19. <<http://www.ivoa.net/internal/IVOA/IvoaVOQL/ADQL-0.91.pdf>>
9. D. J. Rohde, M. J. Drinkwater, M. R. Gallagher, T. Downs, and M. T. Doyle, "Applying machine learning to catalogue matching in astrophysics," *Monthly Notices of the Royal Astronomical Society*, vol. 360, no. 1, 2005, pp. 69-75.
10. A. J. Storkey, C. K. I. Williams, E. L. Taylor, and R. G. Mann, "An Expectation Maximisation Algorithm for One-to-Many Record Linkage, Illustrated on the Problem of Matching Far Infra-Red Astronomical Sources to Optical Counterparts," University of Edinburgh Informatics Research Report (EDI-INF-RR-0318). <<http://www.inf.ed.ac.uk/publications/report/0318.html>>
11. E. L. Taylor, *Ph.D. Thesis*, University of Edinburgh, 2005.
12. E. P. Lim, J. Srivastava, S. Prabhakar, and J. Richardson, "Entity identification in database integration," in *Proceedings of the ICDE*, 1993, pp. 294-301.
13. A. Kementsietsidis, M. Arenas, and R. J. Miller, "Mapping Data in Peer to Peer Systems: Semantics and Algorithmic Issues," in *Proceedings of the ACM Special Interest Group on Management of Data (SIGMOD) Conference*, 2003, pp. 325-336.
14. W. Litwin, A. Abdellatif, A. Zeroual, and B. Nicolas, "MSQL: A Multidatabase Language.," *Information Sciences*, vol. 49, no. 1-3, 1989, pp. 59-101.
15. W. Kent, "The Entity Join," in *Proceedings of the Fifth International Conference on Very Large Data Bases (VLDB)*, Rio de Janeiro, Brazil, 1979, pp. 232-238.

Security Issues in a SOA-Based Provenance System

Victor Tan, Paul Groth, Simon Miles, Sheng Jiang, Steve Munroe,
Sofia Tsasakou, and Luc Moreau

School of Electronics and Computer Science
University of Southampton
Southampton, SO17 1BJ, UK
vhkt@ecs.soton.ac.uk

Abstract. Recent work has begun exploring the characterization and utilization of provenance in systems based on the Service Oriented Architecture (such as Web Services and Grid based environments). One of the salient issues related to provenance use within any given system is its security. Provenance presents some unique security requirements of its own, which are additionally dependent on the architectural and environmental context that a provenance system operates in. We discuss the security considerations pertaining to a Service Oriented Architecture based provenance system. Concurrently, we outline possible approaches to address them.

1 Introduction

The concept and utilization of *provenance* has been recently explored in the areas of Grid and Web Services-based systems and environments. The myGrid project implemented a system for recording the documentation of process in the context of in-silico experiments represented as workflows aggregating Web Services [8]. The GriPhyn Virtual Data System project provides a set of tools for expressing and executing workflows in a Grid environment, where the definitions of the workflows are specified in a high-level workflow language and are stored in a catalog to provide for tracking of provenance of all files derived by the workflow [7]. A trial implementation of an architecture based around a workflow enactment engine was used to demonstrate several mechanisms for handling documentation about the invocation of various Web Services was presented in [17]. Studies are now being conducted towards assessing the use of provenance in large scale applications [3].

Most of the work described however does not explicitly consider *security requirements* revolving around the utilization of provenance. Such an omission will hinder eventual evolution of these systems to industrial strength level, where security is likely to be of primary consideration. This is particularly applicable where provenance is concerned with information of a commercially or legally sensitive nature. Our paper seeks to address this shortcoming by analyzing some

of the security issues that arise within a generic provenance system based on a Service Oriented Architecture (SOA). The primary contributions are:

- Discussing basic security issues within such a system;
- Discussing security issues that arise from scalability concerns.

In the next section, we provide a brief description of the provenance representation we employ; the motivation and justification for this type of representation has been covered extensively elsewhere ([9], [11]). The basic security issues a provenance architecture employing this representation are expounded upon in length in Section 3. Section 4 examines new issues that arise as a result of attending to scalability concerns, and we conclude in Section 6.

2 Provenance Representation

We discuss provenance in the context of the Service Oriented Architecture view [5], which provides the underlying architectural basis for the Web Services / Grid environment. In this view, services are simply considered as components that take inputs and produce outputs, which can be brought together to solve a given problem typically via a workflow that specifies their composition. Interactions with services take place using messages that are constructed in accordance with service interface specifications. In a SOA, clients typically invoke services, which may themselves act as clients for other services; we use the term *actor* to denote either a client or a service in a SOA. We refer to the execution of a workflow that is composed of these interacting actors as a *process*.

We adopt the following definition for provenance within an SOA: the provenance of a piece of data is the process that led to that piece of data. Such a provenance will be represented by some suitable documentation of the process (i.e. workflow execution) that led to the data ([10,9]). It is possible to distinguish between a specific piece of information documenting some step of a process from the whole documentation of the process. We refer to the former as a *p-assertion*, which is essentially an assertion made by an actor pertaining to any aspect of a process. Equivalently, the documentation of a process would therefore consist of a set of p-assertions made by all the actors involved in that process.

The various logical components of an architecture for a SOA-based provenance system is detailed in [12]. For purposes of our discussion in the following chapter, we note that the key feature of this architecture is a dedicated repository for holding only process documentation, which we term a *provenance store*. Actors creating p-assertions about a particular process store them into a provenance store. Actors who wish to answer provenance queries (i.e. queries about the provenance of various data items produced during that process) would submit these queries in an appropriate format to the provenance store, which in turn would return the set of p-assertions holding the necessary data required to answer the query. The answering of queries in the provenance store could be augmented.

3 Security Issues in a SOA Provenance System

We classify our discussion into several main areas of security concern:

3.1 Enforcing Access Control over Process Documentation

An obvious security requirement is the need to control access to process documentation. Access control to data in general is a well studied subject for which many practical techniques already exist. A typical approach in many of these techniques is to identify the sensitivity of information within a specific data item (a database record, for example) and then restrict access to a user base in accordance to their predefined roles or identities. Extending this idea directly to our provenance system by restricting access on the basis of individual p-assertions may not be useful, as p-assertions do not generally provide much useful information if accessed as individual data items. Rather, information about a specific aspect of a process (such as all the services that participated in the production of the final result of a workflow) would be obtained by processing the data contained within an appropriate aggregation of p-assertions from the entire set of p-assertions that constitute the process documentation for the workflow in question.

A useful way then to delineate access control boundaries in this example might be to identify different types of provenance related information with differing levels of sensitivity that can be obtained from processing different groups of p-assertions, and then structure access control on the basis of these groups. Since it is likely that a single p-assertion can belong within many groups, there is now the problem that a user without access to a designated group of p-assertions for a specific purpose may still be able to gather together all the constituent p-assertions of this designated group via his or her access to other grouping of p-assertions that inadvertently contains smaller parts of this designated group.

This is a more general problem related to the issue of inference control, i.e. preventing the inferring of information from existing information [18]. Various approaches to address this concern have been proposed within the context of statistical databases, such as perturbing the stored information and query restriction [16]. Provenance complicates the situation because relationship information between the p-assertions is explicitly stored as well, which significantly eases the ability to infer new information from information in existing p-assertions. Hence, existing approaches are likely require modification to tailor them to this environment. A possible solution may involve supporting the specification of access control authorizations at the granularity of these groups and their associated provenance queries. In addition, suitable cryptographic protocols can be used to ensure that users cannot access data within a set of p-assertions returned as a result of a provenance query, unless they have access rights to all the groups that those p-assertions belong to.

We believe that this problem presents a unique angle on access control for data from the perspective of the data being process documentation in a provenance system. More in-depth investigation into this aspect is required if coherent

access control on process documentation and the subsequent provenance related information derivable from it is to be achieved in industrial strength provenance systems.

3.2 Trust Framework for Actors and Provenance Stores

In a large scale distributed environment, actors that create and store p-assertions regarding specific events of interest may not be directly under the control or even known to actors that will eventually use these p-assertions in some manner to answer a provenance query. Signatures provide a way to link actors with p-assertions they create; a methodology is now needed to provide a trustworthiness measure or rating to specific actors and their p-assertions. Ratings could be based on independent third party ratification of the accuracy of the p-assertions or subjective opinions of all potential consumers of p-assertions produced by specific actors. The methodology could also include methods to provide an aggregated measure of reliability of information obtained from processing a group of p-assertions with different levels of associated trustworthiness.

Similar comments are equally applicable to provenance stores; querying actors could elect to establish trust in provenance stores instead and assume that the stores will in turn assume the responsibility of filtering p-assertions from the various actors that send p-assertions to it for storage. There is clearly some work to be done in articulating the various trust models and relationships possible between actors producing and utilizing p-assertions as well as the provenance store holding these p-assertions. Work of this nature could ideally draw on existing extensive work in the area of trust and reputation in agent mediated interactions [19].

3.3 Accountability and Liability for p-Assertions

An important consideration in any provenance system is the accuracy or objectivity of the documentation recorded. In our representation, a p-assertion is a statement about some aspect of a process by an actor. From a more abstract viewpoint, this statement is however only a subjective view of that aspect by an actor. It can be difficult sometimes, if not impossible, to determine how closely this view tallies with actual reality. This is particularly true in our system, where all information about past processes is only obtainable via actor-created p-assertions. With respect to this, it becomes paramount to forge a clear link between an actor and an assertion that it is responsible for. Such a link, which can be provided through digital signatures, ensures that responsibility and corresponding liability is attributable to the correct actor.

Since p-assertions are created within the context of a process that they describe, actors may elect to include metadata within a p-assertion that links it to another p-assertion created by another actor within that context. Incorporation of incorrect metadata in a p-assertion could potentially create a chain of p-assertions that are incorrectly associated, making it difficult or impossible for a querying actor to correctly answer a provenance query. Again, signatures on this metadata ensures responsibility is attributable to the correct actor. We note

that signatures on p-assertions also serve an additional purpose of guaranteeing their integrity and ensuring that no other parties (for example, the provenance store or other intermediary actors that access the p-assertions) change them intentionally or accidentally.

3.4 Sensitivity of Information in p-Assertions

In a basic example, the p-assertion pertaining to a message exchange between two actors would simply contain the contents of that message verbatim. Depending on application domain requirements however, parts of the message may need to be obscured or transformed in some way when they appear in a p-assertion. A good example of this is found in the electronic health care records domain, where privacy requirements mandate that patient identity on health care records be anonymized ([14], [2]) if the information on the record is being utilized for non-diagnostic reasons (for example, to answer provenance questions about a medical process). If p-assertions are utilized in such a context, then certain data items (such as patient identifiers) that are transmitted in cleartext in the original message exchange between actors must be obfuscated in some manner when stored as part of any created p-assertion.

Along similar lines, there may be situations where an actor may want to ensure that certain parts of the p-assertion it creates is only accessible to certain parties. In the simplest case, this can be achieved by ensuring the appropriate access controls are instituted on the provenance store. However, once a p-assertion is retrieved from the provenance store, it is very difficult to control which parties it is subsequently propagated to. If the asserting actor shares a secret key with certain parties, it can elect to encrypt parts of the p-assertion with this key so that only those parties are able to view it.

3.5 Long Term Storage of p-Assertions

Another issue surrounding provenance storage is long term archival of p-assertions. As p-assertions are signed (and possibly encrypted) prior to storage, there will subsequently be a need to verify the signatures or decrypt them when they are extracted for processing. The certificates for the corresponding encryption / signing keys may expire if the storage duration is substantial, and in extreme cases, the underpinning cryptographic algorithms may themselves become outdated. Such issues must be catered for in some way, for example, by having a key archival facility and re-signing / re-encrypting provenance information periodically over the intended storage duration.

3.6 Creating Authorisations for New p-Assertions

It is likely that p-assertions contain or are derived in some fashion from an existing piece of data in the system. For example, an actor with access to a database may send a message containing an item from that database to another actor. This item is likely to have certain access control restrictions enforced upon it within the security domain of the database in question. When a p-assertion

is created for the transmitted message and recorded to the provenance store, appropriate authorisations must now be established for this new entry to ensure that any future access to it is in accordance with the security policies of the provenance store. Such authorisations may be articulated in the form of access control at the level of groups of p-assertions, as discussed in Section 3.1.

In many cases, it is useful to relate the authorisation for the newly recorded p-assertion in some way to the access control restrictions on the original database item that the p-assertion is based upon. This effectively allows for a more flexible specification of authorisations on p-assertions by taking into account information other than that found in statically predefined security policies on the provenance store. A possible approach towards this end is for an actor to submit additional information along with the p-assertion to be stored. This additional information would be provided by the actor and can then be utilised in an automated manner by the provenance store to generate appropriate authorisations for the new p-assertion.

On the issue of relating authorisations of p-assertions to the authorisations of the data that the p-assertions are based on, we note that an interesting situation may sometimes arise where a more stringent level of access control is mandated on the p-assertions themselves rather than the original data. As an example, consider a bioinformatics domain, where a new drug might ostensibly be designed through some dynamic, unplanned novel application of a standard workflow involving publicly accessible data. In such an instance, the exact sequence and logic of the workflow itself (which can be reconstituted from its provenance) becomes more valuable than the actual data used in the workflow, hence necessitating tighter access controls on it.

3.7 Summary

The first security consideration (Section 3.1) we believe is unique to process documentation intended for provenance purposes; data intended for generic processing is unlikely to have such a requirement. The remaining considerations however are likely to be applicable as well when considering the securing of access to data in the general case. The last two considerations (Section 3.5 - 3.6) are additional enhancements to a provenance security architecture that already adequately addresses the core concerns of access control and non-repudiation. They are not intended to further secure the system, but rather to extend flexibility in the enforcement of security: always an important consideration towards increasing the acceptance and adoptability of security measures.

4 Scalability Related Security Issues

So far our discussion has revolved around the notion of a centralized provenance store, but in practice this will inevitably be distributed for the usual reasons of scalability: the elimination of a central point of failure, the spreading of demand across multiple stores and the ability for stores to exist in different network areas.

In such a situation, related p-assertions (such as p-assertions from two actors pertaining to a message exchange between them) could be recorded in different provenance stores. Actors may then record pointers or links to other provenance stores additionally with or as part of the p-assertions in order to provide a trail for interested parties to retrieve related p-assertions. Such links must again be made attributable to actors through signatures, with a similar motivation as well. Distributed provenance stores may exist in different security domains; hence parties that are recognized and authorised for specific actions on a provenance store in one domain may be unrecognized or be granted different access levels in a provenance store of a different domain. In this instance, a federated identity management infrastructure must be operated and installed in order to permit the authorised parties to follow the trail of links and retrieve all relevant distributed p-assertions.

On a similar theme, if p-assertions themselves are copied or moved between stores that are located in different security domains (for example, in staging of data or for load distribution purposes), the access control restrictions on them in their new destinations needs to be defined. In the simplest case, the newly moved or copied p-assertions retain the same access control restrictions that were associated with them in their original domain and the federated identity infrastructure will function to ensure that any newly introduced identities are recognized appropriately in the correct domain.

5 Related Work

The issues of access control, authorization, integrity and privacy within the context of generic databases is well known and numerous approaches have been proposed and implemented [15]. Within the area of data mining, the issue of maintaining user privacy has become paramount and a large amount of work is ongoing in this area to develop more efficient techniques towards this end [1,6]. Related work explicitly investigating provenance-related security issues is however still relatively scarce. In [4], an abstract security model was developed by identifying generic security relevant attributes based on user requirements across a large range of application domains. The myGrid project [20] also investigated security issues and solutions, but in a manner that was highly application dependent. Specific implementation details of a secure annotation service utilizing Grid and Web Services-centric security technologies is discussed at some length in [13].

Our work is pitched at an intermediate level between an abstract model and a concrete implementation. In particular, the issues that we raise are couched specifically within the context of a SOA-based provenance system. In discussing these issues, we also note which ones are potentially more ‘provenance-centric’ and which ones resemble existing database security issues.

6 Conclusion

In this paper, we provide a representation for provenance in a SOA. We then proceed to describe some of the basic security issues pertaining to provenance in such

an environment and possible ways of addressing them. The issue of enforcing accessing control over process documentation represents a unique challenge that potentially distinguishes security considerations for accessing provenance information from that of a more generic data store. Other issues such as developing a trust framework for actors and provenance stores, establishing liability for creation of p-assertions, sensitivity of information in p-assertions as well as their long term storage considerations are more representative of conventional data security concerns.

The notion of scalability is introduced and the additional approaches required to address the new security considerations that arise as a consequence are discussed. We note that all of these security issues have not been explored in depth here; they represent possible pointers to future research on security in provenance systems which is necessary to create industrial strength systems.

Acknowledgements

This research is funded in part by the EU Provenance project as part of the European Community's Sixth Framework Program (IST511085).

References

1. R. Agrawal and R. Srikant. Privacy-preserving data mining. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, Dallas, Texas, 2000.
2. Sergio Alvarez, Javier Vazquez-Salceda, Tamas Kifor, Laszlo Z. Varga, and Steven Willmott. Applying provenance in distributed organ transplant management. In *this volume*, 2006.
3. Miguel Branco and Luc Moreau. Enabling provenance on large scale e-science applications. In *this volume*, 2006.
4. Uri Braun and Avi Shinnar. A security model for provenance. Technical report, Harvard University, 2002.
5. Steve Burbeck. The tao of e-business services. Technical report, IBM Software Group, October 2000.
6. C. Clifton and D. Marks. Security and privacy implications of data mining. In *Proceedings of the ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery*, 1996.
7. I. Foster, J. Voeckler, M. Wilde, and Y. Zhao. Chimera: A virtual data system for representing, querying and automating data derivation. In *Proc. of the 14th Conf. on Scientific and Statistical Database Management*, July 2002.
8. M. Greenwood, C. Goble, R. Stevens, J. Zhao, M. Addis, D. Marvin, L. Moreau, and T. Oinn. Provenance of e-science experiments - experience from bioinformatics. In Simon J Cox, editor, *Proc. UK e-Science All Hands Meeting 2003*, pages 223–226, September 2003.
9. P. Groth, M. Luck, and L. Moreau. Formalising a protocol for recording provenance in grids. In *Proc. of the UK OST e-Science second All Hands Meeting 2004 (AHM'04)*, Nottingham, UK, September 2004.

10. Paul Groth, Michael Luck, and Luc Moreau. A protocol for recording provenance in service-oriented grids. In Teruo Higashino, editor, *Proceedings of the 8th International Conference on Principles of Distributed Systems (OPODIS'04)*, volume Lecture Notes in Computer Science, pages 124–139, Grenoble, France, December 2004. Springer-Verlag.
11. Paul Groth, Simon Miles, and Steve Munroe. Principles of high quality documentation for provenance: A philosophical discussion. In *this volume*, 2006.
12. Paul Groth, Simon Miles, Victor Tan, Sheng Jiang, Steve Munroe, Sofia Tsasakou, and Luc Moreau. Architecture for provenance systems. Technical report, University of Southampton, February 2006.
13. Imran Khan, Ronald Schroeter, and Jane Hunter. Implementation of a secure annotation service. In *this volume*, 2006.
14. Tamas Kifor, Varga Laszlo, Sergio Alvarez, Javier Vazquez-Salceda, and Steven Willmott. Privacy issues of provenance in electronic healthcare record systems. In *Proc. 1st Workshop on Privacy and Security in Agent-based Collaborative Environments (PSACE 2006)*, *5th International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS 2006)*, Japan, May 2006.
15. T.F. Lunt and E.D. Fernandez. Database security. *SIGMOD RECORD*, 19(4):90–97, 1990.
16. N.R.Adam and J.C.Wortmann. Security-control methods for statistical databases: A comparative study. *ACM Computing Surveys*, 21(4):515–556, December 1989.
17. M. Szomszor and L. Moreau. Recording and reasoning over data provenance in web and grid services. In *Int. Conf. on Ontologies, Databases and Applications of Semantics*, volume 2888 of *LNCS*, 2003.
18. S. R. Wiseman. On the problem of security in database. In D.L.Spooner and Landwehr, editors, *Database Security III*, pages 301–311, North Holland, 1990. Elsevier Science Publishers.
19. H.C. Wong and K.Sycara. Adding security and trust to multi-agent systems. In R.Falcone, C.Castelfranchi, Y.H. Tan, and B.Firozabadi, editors, *Workshop on Deception, Fraud and Trust in Agent Societies: Proceedings of the 3rd International Conference on Autonomous Agents*, Seattle, Washington, 1999. ACM Press.
20. J. Zhao, C. Goble, M. Greenwood, C. Wroe, and R. Stevens. Annotating, linking and browsing provenance logs for e-science. In *Proc. of the Workshop on Semantic Web Technologies for Searching and Retrieving Scientific Data*, October 2003.

Implementing a Secure Annotation Service

Imran Khan, Ronald Schroeter, and Jane Hunter

The School of ITEE
The University of Queensland,
St Lucia, Queensland, Australia
{imrank, ronalds, jane}@itee.uq.edu.au

Abstract. Annotation systems enable “value-adding” to digital resources by the attachment of additional data in the form of comments, explanations, references, reviews and other types of external, subjective remarks. They facilitate group discourse and capture collective intelligence by enabling communities to attach and share their views on particular data and documents accessible over the Web. Annotation systems vary greatly with regard to the types of content they can annotate, the extent of collaboration and sharing they allow and the communities which they serve. However many applications share the need to authenticate the source of annotations and restrict access to them - in order to protect intellectual property rights or personal privacy. This paper describes a secure, open source annotation system that we have developed that uses Shibboleth [1] and XACML [2] to identify and authenticate users and restrict access to annotations stored on an Annotea [3] server.

1 Introduction

Annotations have long been used as a tool to facilitate collaborative scholarly discourse. They enable users to attach additional material such as comments, notes, queries, assessments, references to resources such as documents, images or datasets. When applied to digital resources shared via the Web, they provide a very powerful collaborative tool.

Currently available annotation systems vary widely with respect to the types of content they annotate, the extent of collaboration and sharing they allow and the communities which they serve [4]. Although they have been successfully applied to domains including education, research, medicine [5] and neuroscience [6] in order to capture and exchange metadata, ideas, opinions and interpretations, evaluation of these applications indicates limitations in existing commercial and prototype systems. Current systems are limited by: lack of responsiveness, use of non-standard proprietary technologies; lack of authentication of the annotations’ creator; limited search capabilities; lack of security mechanisms; inability to reply to/stagger annotations; asynchronous sharing only; support for limited media types; coarse granularity and unstructured annotations (single field, free text only).

The main focus of the work described in this paper is to provide annotation tools for collaborators within eResearch. A critical requirement for such a domain is the need to be able to restrict access to annotations attached to a particular collection of digital resources - to a particular group of trusted colleagues - for reasons of privacy,

confidentiality or protection of intellectual property. This is particularly important within eScience, where the annotation or interpretation of the raw document or data, is often more valuable than the target of the annotation. Also by providing researchers with a robust, reliable security infrastructure, they may be more willing to engage in the exchange of views and ideas – a key to successful inter-organizational collaboration.

The security requirements for annotations involve two levels of protection:

- protecting the annotation server through identity management and authentication;
- protecting individual annotations through the specification of access policies that define permissible types of access (e.g, list, read, delete) by individual users or user groups based on user attributes.

Within this paper we describe an open source implementation of a secure annotation service that we have developed. Our implementation involves the combination and extension of a number of existing open source technologies that are based on open standards:

- Annotea – a Web-based annotation server developed by the W3C [7];
- Shibboleth – an Internet2 middleware initiative that enables identity management and secure access to Web resources shared amongst multiple organizations [1];
- XACML (eXtensible Access Control Markup Language) – XML-based language for defining and enforcing access control policies [8].

The remainder of this paper describes in detail the secure annotation system that we have built. Section 2 describes previous related activities in the development of annotation systems and security mechanisms. Section 3 describes the overall architecture of our system and its main components. Section 4 illustrates the user interface and system functionality. Section 5 provides an evaluation of the system and describes future work. Section 6 provides a brief conclusion.

2 Background and Previous Work

Significant prior work has been carried out on both web-based annotation systems and on identity management and role-based access control. Rather than re-invent the wheel, we carried out an analysis of existing systems to determine if any currently available solutions satisfied our needs and hence could be integrated, refined or extended.

2.1 Existing Annotation Systems and Annotea

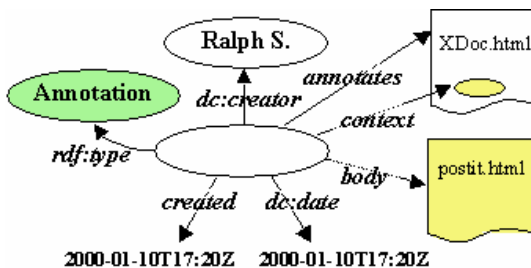


Fig. 1. Annotea's Annotation Schema [5]

A survey of current Web-based annotation systems [4] reveals that they vary in the way in which annotations may be attached, the way in which they are presented and in the access control mechanisms. None of the surveyed systems provide the kinds of fine-grained access control

mechanisms that is required by collaborative teams of scientists engaging in eResearch.

Through earlier work [9] we identified Annotea [7] as an ideal approach for implementing an annotation server. Annotea is a Web-based annotation system that uses Resource Description Framework (RDF) [10] to model annotations as a set of statements or assertions. These annotations are stored in a HTTP enabled server, which enables clients Annotea enabled clients [11] [12] to query, update, post, delete and reply to annotations. Currently there are two publicly available implementations e Annotea servers: Zope [9] and W3C Perllib[13]. Figure 1 illustrates the RDF annotation schema used to describe various properties of an annotation including its author, title, date of creation, body and context. Annotea's use of open W3C standards such as RDF, XPointer, XLink and HTTP makes it possible to easily adapt or extend the scheme. Annotea can also be easily extended to allow for the annotation of media types other than text. For example Vannotea [14] extends Annotea to enable the annotation of videos. Vannotea and applications similar to it clearly identified the need to further extend the Annotea to enable fine-grained access control to annotations [15].

2.2 Identity Management and Shibboleth

Harris et. al. generated a comprehensive report describing access management (AM) systems used in the UK Higher Education sector [16]. The most prominent systems identified included: Microsoft's Passport, Liberty Alliance, WS-Security, PAPI, Athens and Shibboleth. Of the six systems, only three are targeted at the higher education domain, while the other three (Passport, Liberty Alliance and WS-Security) are primarily focused on business-centric solutions. Morgan et al [17] describe Shibboleth as "an open-source system that extends Web-based applications and identity management for secure access to resources among multiple organizations." Shibboleth is based upon a number of open standards including HTTP, XML, XML Schema, XML Signature, SOAP (Simple Object Access Protocol). In particular it is dependent on:

- SAML [18] Security Assertion Markup Language for the exchange of assertions between the Identity Providers and Service Providers
- eduPerson [19] – an EDUCAUSE initiative to define a standard set of person attributes in higher education environments

2.3 XACML

As Lorch et al [8] explains, Shibboleth lacks a dynamic and distributed approach to access control. XACML enables us to address these issues. XACML (eXtensible Access Control Markup language) is an XML-based language used to describe general purpose access control policies and also an access control decision request/response language [18]. XACML also specifies the structure and syntax of the requests and responses. Requests are composed of attributes associated with the requestor, resource being accessed and the action being performed.

3 System Architecture and Implementation

Figure 2 illustrates the overall architecture of the system. The diagram highlights the two key components of the Shibboleth architecture: Identity Provider (IDP) and Service Provider (SP).

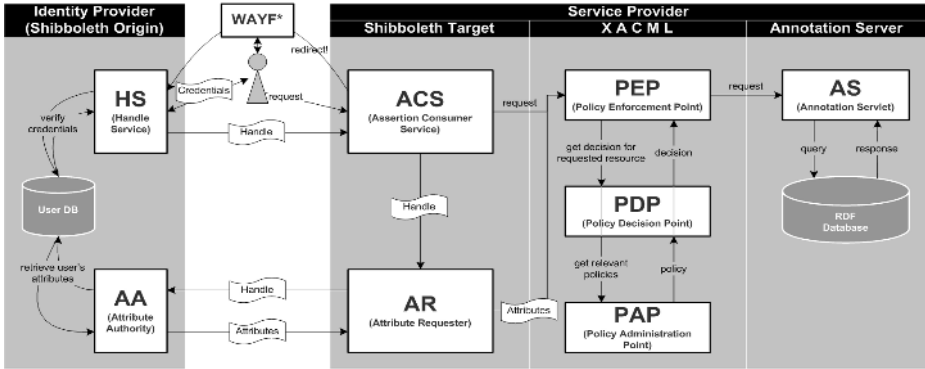


Fig. 2. System Architecture

The annotation server within Figure 2 is part of the SP, and may be located on any of the organizations/universities that are part of the federation. Figure 2 illustrates a simplified view of a shibbolized annotea transaction. Firstly Shibboleth is responsible for authenticating a user and retrieving the users' attributes from the requestors IDP. These attributes are then passed on to the XACML.

3.1 Server-Side

The Server side consists of four main architectural components: the Annotation and Policy server, XACML module; Shibboleth attributes and the Jena database.

3.1.1 The Annotation and Policy Server

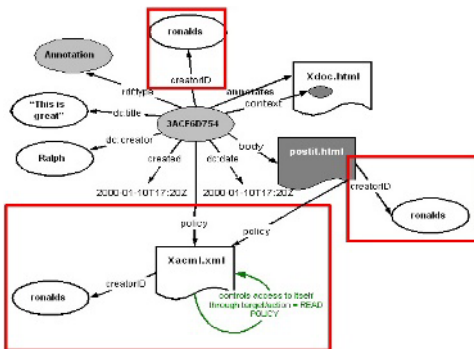


Fig. 3. Extended Annotea Model

The Annotea server has been extended to support the fine-grained access policies in addition to the operations defined by Annotea. Figure 3 illustrates the extensions made to support policies (in red). The first extension is the unique creatorID. The creatorID pro-property is used when making decisions on delete and update operations. The other key extension is the policy object. Uni-que-ly identified XACML policies are stored within the RDF

repository. Objects are linked to particular policies through their policy property – which is specified by a URL. This approach has the benefit of enabling multiple annotations to use the same policy. If a policy is modified, the changes will effect all those annotations associated with the policy.

3.1.2 XACML Module and Policies

This module is responsible for implementing the *Role Based Access Control* functionality and is based upon Sun’s XACML implementation [2]. It makes decisions on whether a particular request is permitted based upon the role/attributes of the person making the request. There are three types of actions permissible on annotations by users other than the creator:

- LIST – viewing of annotation metadata (e.g., author, creation date, etc.)
- READ – viewing of the annotation body.
- READ_POLICY – viewing of the annotation policy.

Figure 4 illustrates an example policy and request. Each **Policy** consists of a set of **Rules** related to whether a specific operation is permitted by a particular Subject. The Subject is described by a set of attributes which identify the credentials of a particular user e.g. affiliation, role. In Figure 4a, members of the Staff “Group” have an attribute *eduPersonAffiliation* equal to “staff”. In the example, staff are permitted to perform all three operations on annotations whilst students are denied access to all three. Given the example policy in Figure 4a, the example request (in Figure 4b) - that a student to be permitted to read policy 123, will be denied. It is important to note that although XACML policies provide much more expressiveness than we provide, we have deliberately kept the user interface (Figure 6) simple so that end users can create policies themselves.

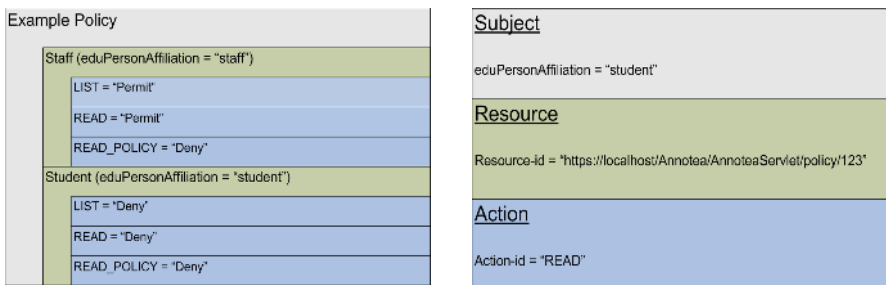


Fig. 4. (a) Example Policy and (b) Example Request

The XACML module is implemented through three steps. The first step involves gathering attributes about the requester from the requestor’s IDP through Shibboleth’s SAML assertions. Using these attributes, an XACML request is created – it specifies the action to be performed, the resource requested and the attributes of the requestor. The second step involves locating the policy associated with the resource being requested by querying the RDF repository for a policy with a given URL. Thirdly the

retrieved policy is compared with the request and Sun's XACML implementation generates an XACML response specifying whether the request is permitted or denied.

3.1.3 Shibboleth (SAML) Attribute Assertions

The annotation server depends on Shibboleth to provide the necessary eduPerson attributes about a requestor. These attributes are used by the XACML module to make an access control decision. Shibboleth itself is a complex architecture and details are available from [20]. Each site within a Shibboleth federation consists of either/both an origin (identity provider) and target (service provider). In terms of our annotation system, a user's attributes are provided by their origin, which stores them in an LDAP server. The annotation server is hosted on the target site and is accessible to members of organizations that are part of the federation and have sufficient access privileges to the annotation server.

3.1.4 Jena Database

Jena [21] provides an API to an RDF repository and in the context of this system is responsible for enabling the storage and interfacing of data – including annotations, policies and annotation bodies. The Jena API also enables us to search the annotations - via the creator, date, language and in_reply_to fields.

3.2 Client-Side

The user's client side application is responsible for the user interface that enables: the retrieval and display of annotated web resources; display, search and browsing of annotations the creation, editing, deletion and attachment of annotations to Web resources; the creation, editing and attachment of access policies to annotations.

Although a number of client-side annotation tools exist for annotating Web resources (Amaya [12], Annozilla [11], and Vannotea [14]) none of these provide an interface suitable for specifying XACML access policies. Consequently we developed our own client-side application using .NET to allow the display and editing of annotations and policies.

4 The User Interface

For testing and illustrative purposes, we used the ePrints archive at the University of Queensland. Figure 5 shows the user interface after an authenticated user accesses the annotation server and retrieves a particular annotated publication. The annotations are displayed in the top left-hand frame, the details of a selected annotation are in the bottom left-hand frame and the publication is displayed in the right hand frame.

Figure 5 also illustrates the user interface for creating and attaching an annotation. We have extended Annotea to support structured annotations that contain a number of fields including hyperlinks, files, free text or controlled vocabularies.

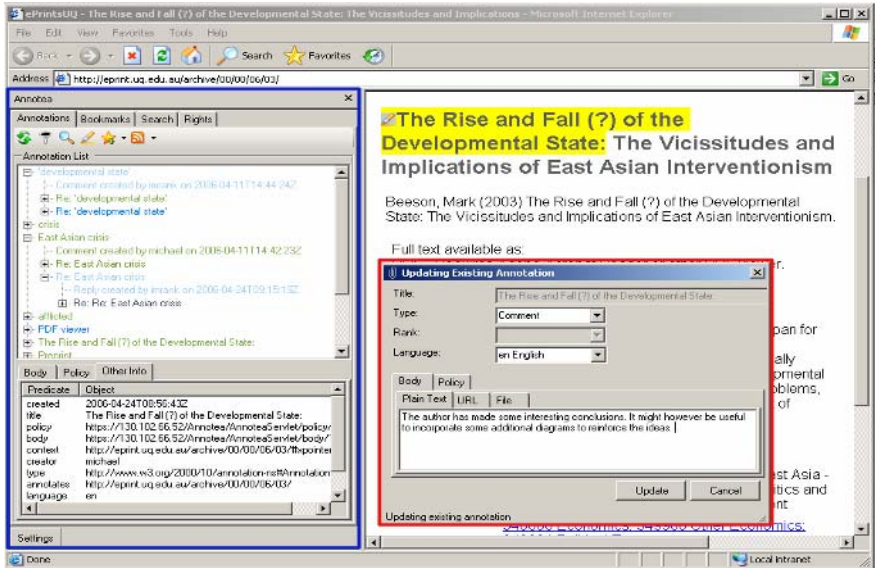


Fig. 5. User Interface showing Sidebar with threaded replies and dialog box for creating annotations

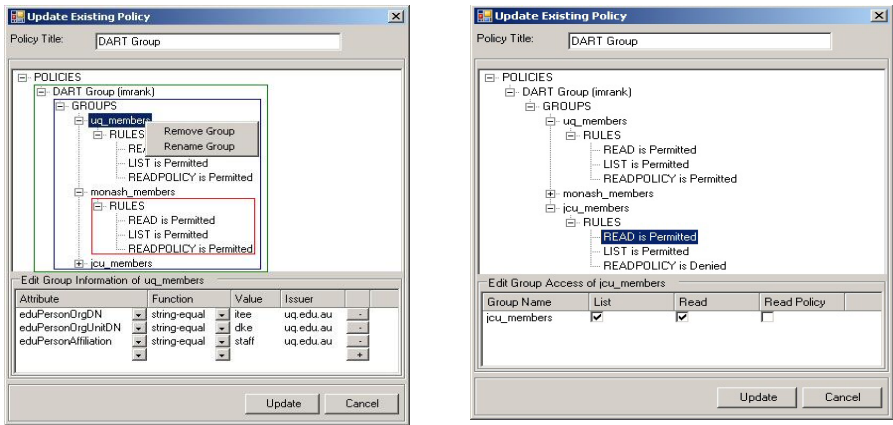


Fig. 6. User Interface for defining Policy (a) Groups and (b) Rules

Figure 7 shows the user interface developed for defining groups and policies. Firstly it enables ‘Groups’ to be defined by sets of common attribute values. In Figure 8a, the group *uq_members* is defined as users with (*eduPersonAffiliation* = *staff*, *eduPersonOrgUnitDN* = *dke* and *eduPersonOrgDN* = *itee*) where the attributes are issued by *uq.edu.au*. The policy in Figure 9a has three groups - *uq_members*, *monash_members* and *jcu_members*. The second part of policy definition involves defining access rules for each of the groups. In Figure 10b, the Group *jcu_members*

are permitted to *List* and *Read* annotations, but not *Read Policy*. This interface makes it easy for users to define new groups, modify/remove existing groups and define/modify policies.

5 System Evaluation and Future Work

5.1 System Evaluation

To date, system evaluation has comprised thorough unit and system testing. This involved testing the creation, editing and deletion of policies and annotations. We also tested policy enforcement by logging on as users with different attributes and modifying attributes directly in the LDAP directory. In all cases the annotation server behaved as expected. However the testing phase did reveal some problematic issues. These included:

- Allowing the deletion and update of annotations can lead to ‘hanging references’ where replies refer to annotations which have been updated or deleted.
- The use of URLs to identify policies enables them to be re-used and applied to multiple annotations. However this may cause problems when a policy referred to by multiple annotations is updated/deleted.

5.2 Future Work

Aspects of this work that would benefit from further investigation include:

- Thorough user evaluation: detailed usability studies are required to acquire user feedback and determine functional requirements of user groups as well as improve, refine and extend the system.
- Reduce reliance on Shibboleth: approaches other than Shibboleth will enable the annotation server to be used outside of the Higher Ed sector.
- Annotation of PDF files and spreadsheets: the popularity of publishing scholarly information in PDF format and storing scientific data in spreadsheets indicates an increasing demand to be able to annotate data in proprietary formats.
- Access policies based on document attributes: it would be interesting to investigate policies that are based on attributes of the digital resources or their annotations.
- Complex querying: the integration of SPARQL to allow more complex queries over the annotation server while enforcing access constraints.
- Post-processing of annotations: currently authors are notified via RSS when replies to their annotations are made. Other examples include the Multivalent Browser [22] which can review and incorporate suggested changes within documents.
- Ontology-based annotations or the annotation metadata could be used to process annotations and automatically classify/rank annotations and resources.
- Scalability: further investigation is required to determine how the system performs as the number of annotations, access policies and users grows.

6 Conclusions

This paper describes a secure annotation service that we have developed by combining and extending a number of existing open source technologies. Secure, trusted annotation servers are required in many domains including telemedicine, higher education and collaborative eResearch. By providing clinicians and researchers with the necessary support for authenticating the source and protecting the confidentiality and intellectual property of their annotations, they will be more willing to share their views and engage in inter-organizational collaborations with trusted colleagues. Moreover, the modular design and interoperable technologies that we have adopted, makes it easy to quickly adapt the server to a variety of different media types, different domains and different communities.

References

- [1] Internet2, "Shibboleth Project," 2005, <http://shibboleth.internet2.edu/>.
- [2] S. Proctor, Sun Microsystems, "XACML API," 2004, <http://sunxacml.sourceforge.net/>.
- [3] M. Koivunen et al, "Annotea: an open RDF infrastructure for shared Web annotations," in *Proceedings of the 10th Intl conference on World Wide Web*, Hong Kong, ACM Press, 2001
- [4] R. Heck et al, Department of Mathematics and Computer Science, Grinnell College, "A Survey of Web Annotation Systems," 1999, http://www.math.grin.edu/~rebelsky/Annotations/Summer1999/Papers/survey_paper.html.
- [5] M. Lewkowicz et al, "A Web-based Annotation System for Improving Cooperation in a Care Network," in *ICWE Workshops*, pp. 227-239, 2004.
- [6] M. Gertz et al, "Annotating Scientific Images: A Concept-Based Approach," in *14th Intl Conference on Scientific and Statistical Database Management 2002*.
- [7] R. Swick et al, W3C, "Annotea Protocols," 2002, <http://www.w3.org/2002/12/AnnoteaProtocol-20021219>.
- [8] M. Lorch et al, "First experiences using XACML for access control in distributed systems," *Proc of the 2003 ACM workshop on XML security* Fairfax, Virginia ACM Press, 2003.
- [9] Zope, "Zope Annotation Server," 2005, <http://www.zope.org/Members/Crouton/ZAnnot/>.
- [10] D. Brickley and R. V. Guha, W3C, "Resource Description Framework (RDF) Schema Specification 1.0," 2005, <http://www.w3.org/TR/2000/CR-rdf-schema-20000327>.
- [11] M. Wilson, Mozdev.org, "Annozilla (Annotea on Mozilla)," 2000, <http://annozilla.mozdev.org/>.
- [12] I. Vatton, W3C, "Amaya," 1994, <http://www.w3.org/Amaya/>.
- [13] W3C, "Perllib Annotations Server HOWTO," <http://www.w3.org/1999/02/26-modules/User/Annotations-HOWTO>.
- [14] R. Schroeter et al, "Vannotea -A Collaborative Video Indexing , Annotation and Discussion System For Broadband Networks," in *Knowledge Markup and Semantic Annotation Workshop, K-CAP 2003*, Sanibel, Florida 2003.
- [15] J. Hunter et al, "Using the Semantic Grid to Build Bridges between Museums and Indigenous Communities," in *Semantic Grid Applications Workshop*, Honolulu 2004.
- [16] N. Harris et al, "Access Management Report," London School of Economics 2002.

- [17] R. L. Morgan, S. Cantor, W. Hoehn, and K. Klingenstein, "Federated Security: The Shibboleth Approach," *Educase Quarterly*, vol. 27, pp. 12-17, 2004.
- [18] R. Cover, Oasis, "Cover Pages: Security Assertion Markup Language (SAML)," 2005, <http://xml.coverpages.org/saml.html>.
- [19] Directory Working Group (MACE-Dir), Internet2 Middleware Architecture Committee for Education, "EduPerson Object Class Specification (Draft)," 2006, <http://www.nmi-edit.org/eduPerson/draft-internet2-mace-dir-eduperson-latest.html>.
- [20] T. Scavo and S. Cantor, Internet2, "Shibboleth Architecture," 2005, <http://shibboleth.internet2.edu/docs/internet2-mace-shibboleth-arch-conformance-latest.pdf>.
- [21] B. McBride, "Jena: A Semantic Web Toolkit," *IEEE Internet Computing*, vol. 6, pp. 55-59, 2002.
- [22] A. P. Thomas et al, "The multivalent browser: a platform for new ideas," in *Proceedings of the 2001 ACM Symposium on Document engineering*, Atlanta, Georgia, USA, ACM Press, 2001.

Performance Evaluation of the Karma Provenance Framework for Scientific Workflows

Yogesh L. Simmhan, Beth Plale, Dennis Gannon, and Suresh Marru

Indiana University, Bloomington IN 47405, USA
{ysimmhan, plale, gannon, smarru}@cs.indiana.edu

Abstract. Provenance about workflow executions and data derivations in scientific applications help estimate data quality, track resources, and validate *in silico* experiments. The *Karma provenance framework* provides a means to collect workflow, process, and data provenance from data-driven scientific workflows and is used in the Linked Environments for Atmospheric Discovery (LEAD) project. This article presents a performance analysis of the *Karma* service as compared against the contemporary *PReServ* provenance service. Our study finds that Karma scales exceedingly well for collecting and querying provenance records, showing linear or sub-linear scaling with increasing number of provenance records and clients when tested against workloads in the order of 10,000 application-service invocations and over 36 concurrent clients.

1 Introduction

Data-driven scientific investigations often follow a dataflow pattern where data progresses through a number of processes as they are transformed, fused, and used in complex models. Services provide an abstraction to access these processes through well-defined interfaces and allow applications to be modeled as workflows that capture the invocation logic. *Process provenance*, collected about the workflow, describes the service invocations during a workflow's execution and enables tracking of workflows and services in collaborative environments [4,15]. In data-driven applications, provenance about the data involved in the workflow is critical to understanding its results. *Data provenance*, the derivation history of derived data, includes the service and its parameters that contributed to the data creation, and is valuable to determine the origin and quality of a particular derived data, and for its discovery and reuse in other workflows [15].

The *Karma provenance framework* [16] records uniform and usable provenance metadata for scientific workflows that meets the domain needs of the Linked Environments for Atmospheric Discovery (LEAD) project [14] while minimizing the performance overhead on the workflow engine and the services. It collects two forms of provenance: *process provenance*, also known as process-oriented provenance or workflow trace, describes the workflow's execution and associated service invocations, and is used to monitor the workflow progress and mine it for results validation; and *data provenance*, which provides complementary metadata about the derivation history of data products in the workflow, including

the services that create and use it, and the input data transformed to generate it, and forms the basis for quality-oriented data product discovery [17].

Karma is used to collect provenance from meteorology workflows in LEAD, where hundreds of simultaneous users are expected to run workflows and query for provenance at any given time. This article describes our empirical evaluation of the Karma provenance framework in meeting the needs of the LEAD project to record and retrieve provenance documents for different workflow loads and with concurrent clients. We present our experimental results alongside the performance results for *PReServ* [8,9], a comparable service for recording provenance assertions, for equivalent workloads.

Several provenance frameworks have emerged in the past several years [19,18,12,9,7,3,6] that have defined the provenance needs for e-Science and have applied the systems to different scientific domains. Surveys on provenance have compared these systems based on a meta-model for provenance [4] and through the use of a taxonomy [15]. While some of the provenance systems have presented a preliminary evaluation of the performance of their systems [8], none have undertaken a detailed comparative performance study of their provenance framework as we have in this article. Such a study, based on multiple workflow and query loads, is necessary to determine the overhead of collecting provenance, and the costs for querying and using provenance metadata under different application scenarios. A comparative evaluation also provides additional context to interpret the results. PReServ was selected as the provenance system compared against due to several reasons. PReServ is similar to Karma in that it is a stand-alone provenance service independent of the workflow or service environment, and it is motivated by the provenance requirements for scientific experiments. It also provides the ability to store metadata annotations as part of provenance, that allows us to record provenance akin to that captured by Karma. Lastly, it is a contemporary system that is being actively developed and used.

The rest of the article is organized as follows: in Section 2, we briefly describe the architecture of Karma; in Section 3, we discuss the hardware and software deployment used in the experiments; Section 4 describes the experiments for collecting provenance and their results, and Section 5 does likewise for querying provenance; Section 6 discusses the results of the experiments, and, finally, Section 7 presents our conclusion and future work.

2 Architecture

Karma collects provenance for data-centric workflows in a service oriented architecture. Such workflows are composed of services connected as directed graphs, with each service passing one or more data product as input to the service it is connected to, thus forming a dataflow. The workflow is orchestrated by an execution engine, and execution takes place at three levels. At the *workflow level*, the workflow engine invokes services with appropriate parameters, in the order prescribed in the workflow graph. At the *service level*, the service that receives the invocation from the workflow engine initiates action by executing a method

or launching an application. At the *application level*, the actual task corresponding to the service invocation is performed. Each workflow, service, application, and data product used in the workflow is identified using a globally unique ID.

Karma uses the notion of activities [16] that take place at different levels of a workflow’s execution, in space and in time, to collect provenance (Fig. 1). The key activities are *Workflow-Started* and *-Finished*, generated by the workflow identified by its *Workflow ID*; *Application-Started* and *-Finished*, generated by the invoked service (or application) identified by its *Service ID*; and *Data-Produced* and *-Consumed*, submitted by the end-application to list the *Data Product ID* of the data it transforms. Based on these activities, Karma builds three forms of provenance documents that can be retrieved: *workflow trace* describes all activities for a workflow’s execution, *process provenance* captures activities for a single invocation of a service or application within the workflow, including its input and output data, and *data provenance* returns the application that created the data product and those that use it, all potentially from different workflows.

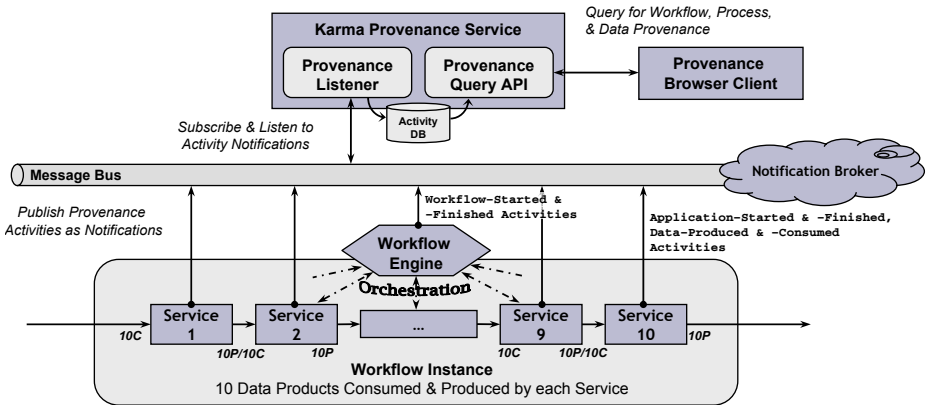


Fig. 1. Architecture of Karma provenance framework. At the bottom, a linear workflow is orchestrated by a workflow engine and publishes provenance activities on the workflow, services, and data products as notifications. A listener in the Karma service subscribes to and receives these notification from the notification broker, and persists them in a database. Upon query by a provenance client, the Karma service constructs and returns the workflow, process, or data provenance graph from the activities.

Provenance is submitted by the workflow components, namely the workflow engine, services, and applications, either synchronously through a web-service call to the Karma service or asynchronously. In the latter case, provenance activities are modeled as XML notifications that are published to a notification broker that the Karma service subscribes to, making it easier for messaging-aware applications to submit provenance. For our experiments, we use the asynchronous approach, with notifications published using the *WS-Eventing* standard [5] through the *WS-Messenger* [10] notification broker implementation. The Karma service persists the activities it receives to a relational database and provides

a web-service API to query for the workflow trace, process provenance, and data provenance. The provenance graphs are constructed on-the-fly from the activities recorded in the database and represented as XML documents with a defined schema. A graphical interface is available to visualize and navigate the provenance graphs.

3 Experimental Setup

The components of both Karma and PReServ services run on identical nodes in a Linux cluster, each node consisting of dual 2.0 GHz 64 bit Opteron processors with 16 GB RAM. Karma service v0.3, WS-Messenger notification broker, and a mySQL database used by Karma run on separate nodes. The PReServ service v0.2.3 is deployed on a single node within a Tomcat 5.0 web-server container and uses the default embedded Java database. Clients that generate and retrieve provenance run on a separate 128-node Linux compute-cluster formed of dual 2.0 GHz Opteron processors with 4 GB RAM. Parallel executions of client programs is managed by a SLURM job manager [2]. All nodes in both clusters are connected by Gigabit Ethernet and use local IDE disks for persistent storage. The services and test applications are written in Java and Jython, and use Sun Java 1.5 JVM as the Java runtime.

4 Collecting Provenance

Workflows, services, and applications in LEAD are written in Java or as Jython scripts which invoke FORTRAN binaries. They are instrumented using the *Notifier* Java library to publish Karma provenance activities as asynchronous notifications. The services in the workflow are usually auto-generated web-services wrappers for applications, to enable their use in a service-oriented architecture [11]. The provenance instrumentation in the case of workflows and services is automated. Applications written by the service providers are manually instrumented using the Notifier library to generate the provenance activities.

In the experiments for collecting provenance, the standard Notifier library is used by all workflow components to generate the Karma activities asynchronously. For evaluating PReServ, the default Notifier implementation is replaced by a thin client that synchronously records provenance with the PReServ web-service using its client library. The following experiments measure the provenance generation overhead for different workflow loads shown in Fig. 2.

In both Karma and PReServ, a single invocation of a service in a workflow generates a set of provenance records. For Karma, the provenance recorded is as follows: one each of *Application-Started* and *-Finished* activities marking the start and end of the invocation, and identifying the workflow, service, and application scope in which the invocation took place; and one *Data-Consumed* or *-Produced* activity for each data product used or created during the invocation, containing the *Data Product ID*, its location, usage or creation time, and its size [16]. In the

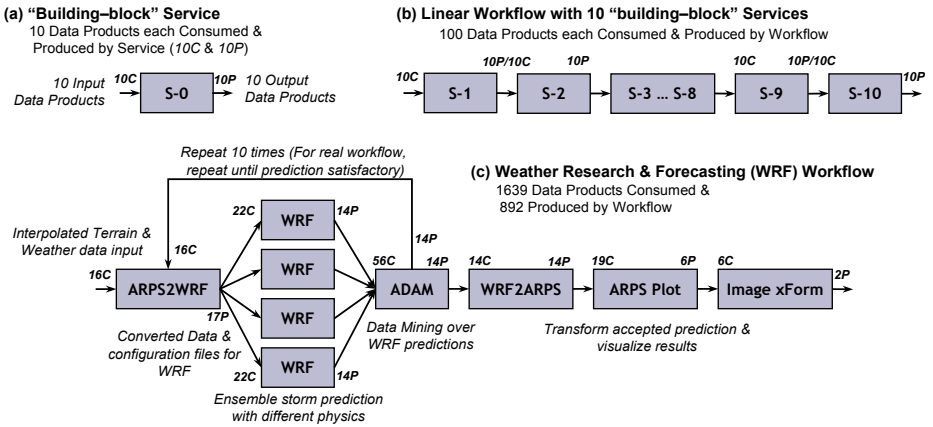


Fig. 2. Types of workflows and services used to generate provenance for experiments. The rectangles denote the service in the workflow and the arrows are the invocation order or data flow. The arrows are labeled with the number of data products that flow between services.

case of PReServ, a service invocation generates three types of provenance assertions: one **Interaction** assertion that establishes the occurrence of the invocation by providing the source and sink of the service invocation along with a unique ID for that interaction; one **Actor State** assertion with the list of data products produced and consumed during that interaction, identified by their *Data Product IDs*; and two **Relationship** assertions that respectively associate the **Interaction** assertion with the produced and consumed data present in the **Actor State** assertion [8]. PReServ optionally allows for recording the SOAP message for the service invocation in the assertions but this is not used in any of the experiments.

4.1 Single Service

The first experiment evaluates the performance of recording provenance for a single service as the number of service invocations and the number of provenance records already present in the provenance service increase. The simple “building-block” service shown in Fig. 2(a) acts as a client that generates provenance about its invocation. It is modeled as a Java application that takes 10 data products as input and generates as many as output, doing no computation or I/O operations other than record provenance. This building-block service is repeatedly invoked from a test harness running in the same JVM, and, for each invocation, the service records a set of provenance records describing its invocation.

As noted earlier, for this service invocation, Karma generates two activities marking the start and end of the application and 20 activities on the data products used and created. These map to four notifications that are published. While the **Application-Started** and **-Finished** activities are published as two individual notifications, the set of 10 **Data-Consumed** and 10 **Data-Produced** activities are batched as two notifications. PReServ generates one **Interaction** assertion, one

Actor State assertion, and two Relationship assertions, for a total of 4 provenance assertions. Karma and PReServ represent their provenance activities and assertions as XML documents, and for this single service invocation, the total size of all XML provenance documents recorded in both cases is about 10KB.

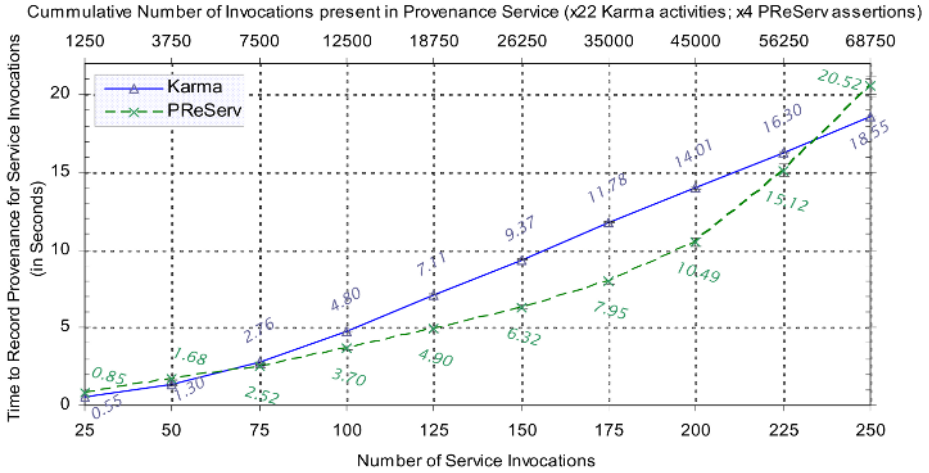


Fig. 3. Total time to record provenance (Y axis) for “building block” service as the number of service invocations per trial (bottom X axis) and the number of provenance records present with the provenance service (top X axis) increase. Each trial is averaged over 50 iterations. Barring two points, the standard error for all averages are under 5%; it is under 10% for all points.

Figure 3 shows the total time taken to record the provenance for all service invocations in a trial as the number of invocations in a trial increase from 25 to 250 along the bottom X axis. As the trials progress, the records accumulating with the provenance services are shown in the top X axis. As the number of invocations rise, Karma shows a linear behavior in provenance recording time, averaging 74 ms per invocation for 250 invocations when there are over 1.5 million provenance activities (68,750 invocations \times 22 activities per invocation) in the Karma service. This also exhibits a sub-linear recording time against the number of records present in the service. Publishing provenance activities asynchronously as notifications insulates the clients from potential fluctuations in the backend store, although for the workloads that are used, synchronous recording of activities shows similar results too.

PReServ shows super-linear trend as the number of invocations increases from 25 to 175. While taking lesser time at 7.95 seconds for 175 invocations compared to Karma’s 11.78 seconds, beyond that the recording time rises quadratically, with 250 invocations taking 20.52 seconds. However, the time increases almost linearly against the number of records present in the service. This indicates that the performance limitation of PReServ is imposed by its backend store used

to persist provenance records. PReServ shreds and stores the XML provenance records into an embedded Java database that allows for easy portability and deployment, but this experiment shows the limitations of such an approach in scaling beyond 140,000 records (after 175 invocations).

4.2 Simultaneous Linear Workflows

This experiment evaluates the scalability of collecting provenance as the number of concurrent clients generating provenance grows. A simple linear workflow composed of 10 building-block services connected linearly is used as shown in Fig. 2(b). This workflow provides a uniform load under which the provenance systems can be compared, with each workflow run generating 220 Karma activities (40 notifications) and 40 PReServ assertions from the 10 services. Each workflow is started simultaneously on multiple hosts as parallel jobs and a workflow controller program invokes the 10 services in sequence within the same JVM and iterates over 50 trials.

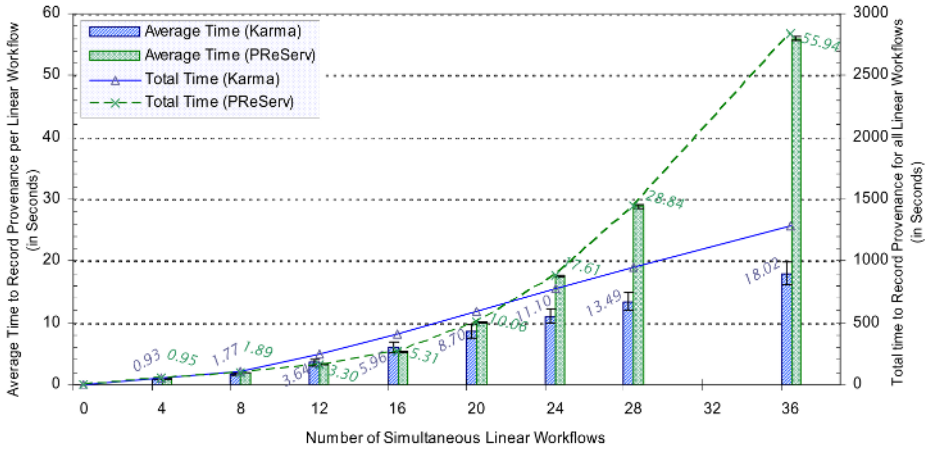


Fig. 4. Times to record provenance for linear workflows with the number of workflows running simultaneously. The left Y axis bar graph shows the average time (over 50 iterations) taken by each workflow to complete as the number of parallel workflows increase along X axis. The data points are labeled with the average time. The right Y axis line graph has the total time for all 50 iterations of each trial to complete and is scaled by 50 times the left Y axis. The standard error for the average workflow time is between 5–15% for Karma and under 3% for PReServ.

Figure 4 shows the time for each workflow run (bar graph on left Y axis) averaged over 50 iterations and the total time for the 50 iterations (line graph on right Y axis) as the number of concurrent workflows increases from 4 to 36 along the X axis. Karma and PReServ show similar performance until 8 concurrent workflow clients, taking an average of less than two seconds per workflow with 8 workflows. As the parallelism increases from 12 to 20, PReServ outperforms

Karma but begins to display super-linear behavior. Karma shows good scalability by maintaining a linear trend throughout, averaging 18 seconds per workflow with 36 parallel clients, compared to 56 seconds for PReServ, which is at best a quadratic trend. This may partially be attributed to the increase in the number of records stored in PReServ, as observed earlier. The standard error for the average workflow time for Karma is between 5–15% while PReServ has more uniform provenance recording time with standard error under 3%.

4.3 Simultaneous Complex Workflows

In order to measure provenance collection performance under realistic workloads, a mesoscale storm prediction workflow from the LEAD project is used in this experiment. Figure 2(c) shows a synthetic workflow involving four parallel Weather Research & Forecasting (WRF) applications that repeat 10 times. The services perform no computation or I/O but generate the same provenance as a real WRF workflow would – 2657 Karma activities (252 notifications) and 252 PReServ assertions from the nine services in the workflow. Such a simulated workflow accelerates gathering of performance results; a real WRF workflow for this experiment configuration requires the use of 320 compute nodes for 1 week. Also, a sample run of the WRF workflow showed high error margins due to I/O contention by the applications, precluding accurate determination of the provenance overhead. In an ideal situation, the computation and I/O time for the real applications should remain constant across concurrent runs, and the synthetic workflow that is used in its stead duplicates such a scenario.

Figure 5 shows the time taken to run the WRF workflow as the number of concurrent workflows increase from 1 to 20. Such a workload is typical in the LEAD system where numerous users simultaneously run such complex workflows [13]. The average time each workflow takes over 25 iterations is shown in the bar graph on the left Y axis and the total time for all workflows to complete 25 iterations is the line graph on the right Y axis while the X axis shows increasing parallelism. As in the previous experiment, Karma achieves linearity while PReServ shows super-linear behavior as concurrency increases. The unsynchronized forks and joins of the four ensemble WRF applications from different workflow instances causes the graph to be relatively flat up to 8 concurrent workflows. In this experiment and the previous, there is a marked increase in the average workflow run time for both Karma and PReServ beyond about 8 parallel workflows. This is likely due to the provenance services reaching a hardware or OS imposed limit, such as network socket availability or bandwidth. The trial for 20 concurrent workflows could not be completed for PReServ since the local disk used by it ran out of disk space. PReServ takes 8 GB to store the provenance records for 16 concurrent workflows while Karma’s MySQL database uses 700 MB – an order of magnitude difference in the storage overhead for the two systems.

4.4 Number of Data Products

Karma stores fine-grained information about the data provenance that requires an activity for each data product consumed or produced by a service. Each data

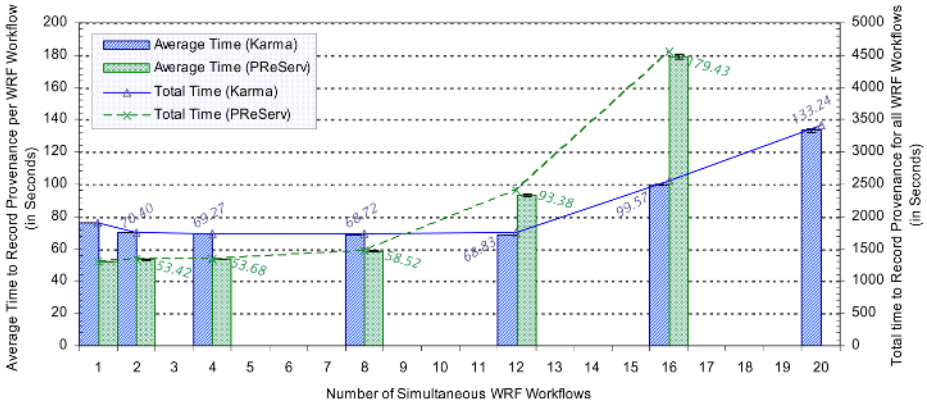


Fig. 5. Average and total time to record provenance for synthetic WRF workflows running in parallel. Left Y axis bar graph represents the average time for each WRF run to complete when averaged over 25 iterations as the number of parallel WRF workflows increases along X axis. The data points are marked with this average time. The right Y axis line graph shows the total time for all 25 iterations to complete, and hence is scaled by 25 times the left Y axis. The trial for 20 concurrent workflows using PReServ could not be completed. The standard error for all averages is below 1%.

product activity contains information such as the *Data Product ID*, creation time, and size, in addition to information about the workflow component that generated it. These allow Karma to natively build the data provenance for any data product involved in various workflows. PReServ does not prescribe any particular metadata to be provided for data provenance and allows any metadata to be submitted as part of its *Actor State* assertion. For the experiments, the *Actor State* assertion for a service carries the essential information about all data products involved in its invocation, namely their *Data Product IDs*.

This experiment estimates the provenance collection overhead as the number of data products involved in a service invocation increases. The single linear workflow shown in Fig. 2(b) is used to generate provenance, but suitably modified so that the constituent services consume and produce progressively increasing number of data products – from 25 to 250 each, for a total of 250 to 2500 data products per workflow.

In Fig. 6, the Y axis shows the average time to record provenance for each workflow as the total number of data products involved in a workflow increase along the X axis. There is negligible difference between the performance of Karma and PReServ for 250 data products used in the workflow. Beyond this, the average time to record provenance using Karma increases linearly with the number of data products, taking 10.77seconds for the workflow involving 2000 data products. This rise correlates with the increase in the size and complexity of the data product batch XML notifications that contain the *Data-Produced* and *-Consumed* activities for each service invocation. PReServ shows a near-constant time for recording provenance as the number of data products increase and this can be attributed to the fact that the minimum information it stores about the

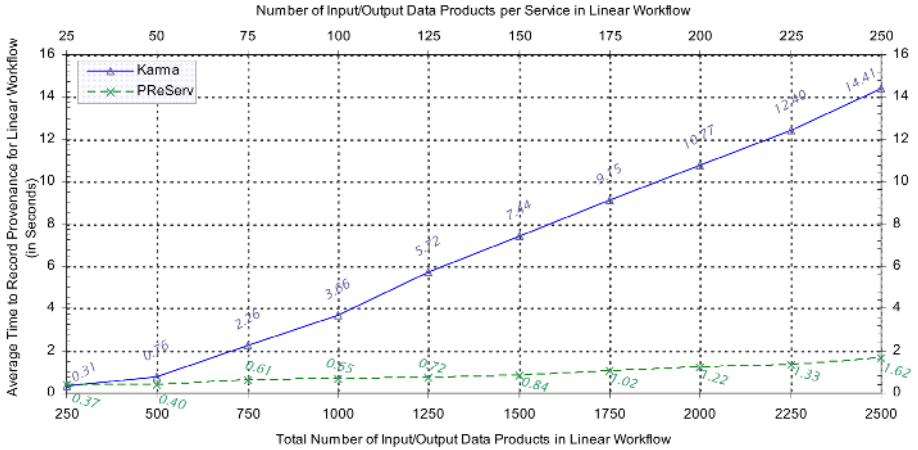


Fig. 6. Average time to record provenance from linear workflow with increasing data products involved in each workflow. The average time over 100 iterations is along the Y axis and labeled at the data points. The lower X axis has the total number of data products involved in the workflow. The upper X axis has the number of data products produced and consumed by each service in the workflow, and is scaled to 10 times the lower X axis since there are 10 services in a workflow.

data products in the Actor State assertion is represented using a simpler XML document. In the LEAD project, 95% of workflows involve less than 250 data products [13] and the performance of Karma for those workflows is comparable to PReServ's. The advantage of being able to record and query data provenance natively offsets the increase in overhead for the other 5% of workflows.

5 Querying for Provenance

Karma builds three types of provenance documents from the activities, namely workflow trace, process provenance, and data provenance. It provides a web-service interface to retrieve these based on the *Workflow ID*, *Service ID*, and *Data Product ID* respectively. The PReServ web-service allows provenance retrieval using *XQueries*, and two simple *XPath* queries that return all Interaction assertions for a given *Workflow ID* and for a given *Service ID* form the equivalent of workflow trace and process provenance queries in Karma. Building data provenance using PReServ requires a more complex query over Interaction, Actor State, and Relationship assertions and is hence omitted for these experiments.

For the following tests, the provenance services are loaded with provenance records for workflow runs and queried for through their web-service APIs from Java clients using the respective client libraries provided by the systems. In the case of Karma, provenance for 1000 linear workflows like those in Fig. 2(b) are loaded, with each service consuming and producing 10 data products. This translates to 10,000 service invocations or 220,000 provenance activities present

in Karma. PReServ is loaded with only 100 linear workflows like those in Fig. 2(b) and corresponds to provenance for 1000 service invocations or 4000 provenance assertions. The factor of 10 difference in the number of workflows recorded with Karma (1000 workflows) and PReServ (100 workflows) is because PReServ is unable to complete queries over 1000 workflows, requiring memory in excess of the 4 GB assigned to it, thus imposing a bound on its query scalability.

5.1 Query Result Size

This experiment measures the query response time as the number of provenance records retrieved increases. For the each of the three provenance types, a single client queries by *Workflow ID*, by *Service ID*, and by *Data Product ID* respectively, making one call to the provenance service per provenance document. For Karma, the queries fetch between 0.01% and 10% of provenance documents, distributed uniformly to prevent any locality advantage. This translates to retrieving between 1–100 workflow traces out of 1000 available, 10–1000 process provenance out of 10,000 available, and 100–10,000 data provenance documents out of 100,000 available. For PReServ, the queries return between 1–10 workflow traces of 100 available, and 1–100 process provenance documents out of 1000 service invocations available. Each query is averaged over 50 trials and the response time plotted as a log–log graph shown in Fig. 7.

The X axis of the plot shows the resultset size from each type of query and the Y axis shows the average response time of the query, both these axes being in the logarithmic scale. All three query types for Karma and both types for PReServ are parallel to the central diagonal on the log–log plot, implying that they all have linear characteristics. But the slopes for the linear equations are markedly different. Karma conservatively takes a factor of 50 lesser time than PReServ for workflow trace queries (e.g. 0.49seconds vs. 26.9seconds for retrieving 10 workflow trace documents) and a factor of 200 lesser time for process provenance queries (e.g. 0.97seconds vs. 209.5seconds for querying 100 process provenance records). The query response time for data provenance is also low for Karma at 55.86seconds for 10,000 data provenance records. The scalability of Karma in responding to queries can be attributed to mapping the provenance activities from an XML schema to a relational schema for storage, and the provenance queries translate to SQL queries that leverage indices present on key fields. PReServ provides an XQuery interface for querying over provenance records and its database does not utilize any indices [1]. This causes all provenance records to be accessed for resolving a query, which also leads to it running out of memory when 1000s of records are queried over.

5.2 Simultaneous Query Clients

This experiment evaluates the scalability of the provenance service as the number of parallel clients querying for provenance records increases from 1 to 48. This is performed only for Karma since PReServ is less optimized for querying and

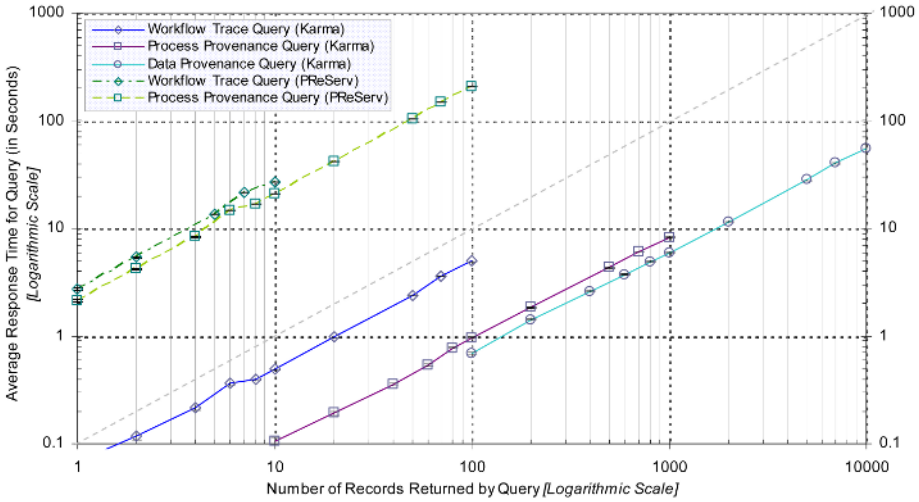


Fig. 7. Query response time to retrieve provenance with increasing number of records returned. The average time to respond to each type of query is along the Y axis and the number of records that the query fetches increases along the X axis. Both axes are in logarithmic scale. The averages times are over 50 iterations; the standard errors for the response times for Karma is under 6% for all but 3 data points, and for PReServ is under 3% for all data points.

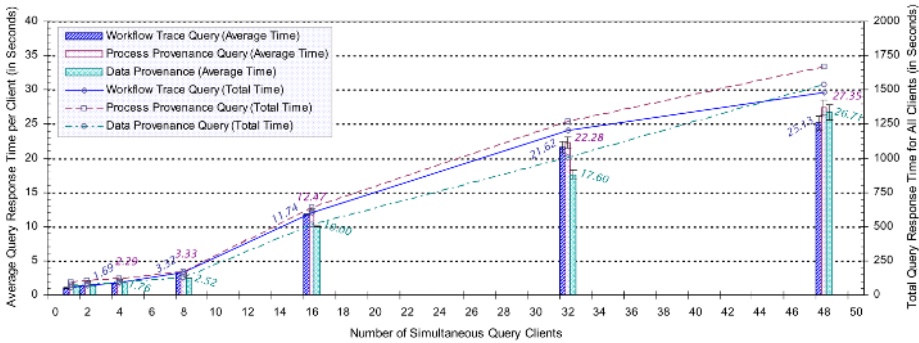


Fig. 8. Query response time to retrieve 20 workflow trace, 200 process provenance, and 200 data provenance from increasing number of concurrent query clients. The average query time over 50 iterations is shown on the left Y axis bar graph, the data points being labeled with this value. The total time to complete the 50 iterations is the line graph on the right Y axis, and is accordingly scaled by 50 times compared to the left Y axis. The X axis shows the increasing number of parallel query clients. The standard errors for all averages are under 4%.

hence results collected for it are less relevant. For the three types of provenance queries, each Karma client retrieves 20 workflow traces, 200 process provenance, and 200 data provenance documents respectively.

Figure 8 shows the average query response time for each query type along the left Y axis as the number of concurrent clients increases along the X axis. The times are averaged over 50 iterations with the total time for all clients to complete the 50 iterations shown on the right Y axis. Karma shows a sub-linear trend as the number of clients increases beyond 8, taking an average of 3.32seconds and 25.13seconds respectively to retrieve the workflow trace from 8 and 48 clients. The results for process and data provenance queries similarly exhibit good scalability. As seen when submitting provenance from concurrent clients, the slope increases when the number of clients goes beyond 8 and may be attributed to an OS or machine threshold being reached.

6 Discussion

The above experiments establish the scalability of Karma for collecting and querying provenance over hundreds of thousands of services and from numerous clients. The workloads used in the experiments are typical in large collaborative scientific projects like LEAD [13]. Provenance collection for Karma shows a linear trend with a low slope for both increasing number of service invocations and for increasing levels of concurrent clients, and performs better than PReServ in these experiments. PReServ shows better characteristics with increasing number of data products, taking constant time unlike Karma which takes linearly time. However, it remains comparably low for Karma in 95% of the use cases for LEAD, that involve less than 250 data products per workflow. Querying Karma for workflow trace, process provenance, and data provenance increases in at most linear time as the number of results retrieved and the number of clients increase, and it uniformly performs better than PReServ.

The difference in performance of Karma and PReServ is due both to design choices and their implementations. Karma and PReServ share several features. They are both stand-alone provenance frameworks and define different types of messages to record provenance, synchronously or asynchronously, from a workflow's execution – Karma using activities and PReServ using assertions. However, PReServ takes an open ended approach to defining provenance assertions, requiring just a few fields to determine the workflow provenance and allowing additional user-defined annotations to be submitted as part of the assertions. While this flexibility may be required for certain applications, it tends to overlap with the functionality provided by existing information services like metadata catalogs and registries for data products and services. Such flexibility may also impose limitations on its implementation, contributing to reduced performance and scalability. Also lacking is inherent support for tracking data provenance, being left to the user to define it as annotations. Karma's activities are less expansive but contain sufficient information to recreate provenance about a workflow run, a service invocation, and the derivation and usage history of a data product. It provides a light-weight and scalable implementation to meet the core needs of recording and querying for these provenance graphs over hundreds of thousands of service invocations and data products. In its current implementation, PRe-

Serv is better suited to record provenance for a smaller number of workflows but with rich XML annotation capabilities, while Karma is effective in meeting more direct and scalable provenance needs in large collaboratories.

7 Conclusion and Future Work

This article evaluates the performance of the Karma provenance framework in collecting and querying for provenance from workflow executions, and finds it to scale well with the size of the workflows and the number of concurrent clients. The workloads used for the experiments are motivated by the requirements of the LEAD meteorology project [13] and is relevant to similar scientific projects. The workloads in themselves form a benchmark to compare and evaluate other provenance systems, and such a comparison is done with the PReServ service.

Karma is currently deployed and being used in the LEAD testbed. Our future work includes evaluating the performance of Karma for real workflow runs and getting usable results for them by suppressing the I/O variations we encountered in the data intensive applications – possibly by the use of local storage instead of network file systems. In addition to visually browsing provenance graphs, we are investigating further ways to apply provenance. Notable among these is on using data provenance as a factor in predicting the quality of data products to assist in data selection and ranking in collaboratory environments [17]. Provenance helps identify applications that produce good or poor quality data as a function of their inputs. In the quality model we propose, this provenance function is one of several metrics used to estimate a quality score for derived data products.

Acknowledgments. This work is supported in part by NSF cooperative agreement ATM-0331480 and NSF grant EIA-0202048. The authors would like to thank Paul Groth from the University of Southampton for helping us deploy the PReServ server, the members of the LEAD team for their support and feedback on our work, and Abhijit Mahabal and Ramyaa Ramyaa from Indiana University for their help in analyzing the empirical data.

References

1. Personal communication with Paul Groth, University of Southampton, 2006.
2. Simple Linux Utility for Resource Management (SLURM) Reference Manual. Technical Report UCRL-WEB-201386, Lawrence Livermore National Laboratory, 2006.
3. Ilkay Altintas, Oscar Barney, and Efrat Jaeger-Frank. Provenance Collection Support in the Kepler Scientific Workflow System. In *IPAW*, 2006.
4. Rajendra Bose and James Frew. Lineage Retrieval for Scientific Data Processing: A Survey. *ACM Computing Surveys*, 37(1):1–28, 2005.
5. Don Box, Luis Felipe Cabrera, Craig Critchley, Francisco Curbera, Donald Ferguson, Alan Geller, Steve Graham, David Hull, Gopal Kakivaya, Amelia Lewis, Brad Lovering, Matt Mihic, Peter Niblett, David Orchard, Junaid Saiyed, Shivajee Samdarshi, Jeffrey Schlimmer, Igor Sedukhin, John Shewchuk, Bill Smith, Sanjiva Weerawarana, and David Wortendyke. Web Services Eventing (WS-Eventing), August 2004.

6. Uri Braun, Simson Garfinkel, David A. Holland, Kiran-Kumar Muniswamy-Reddy, and Margo I. Seltzer. Issues in Automatic Provenance Collection. In *IPAW*, 2006.
7. Juliana Freire, Claudio T. Silva, Steven P. Callahan, Emanuele Santos, Carlos E. Scheidegger, and Huy T. Vo. Managing Rapidly-Evolving Scientific Workflows. In *IPAW*, 2006.
8. Paul Groth, Michael Luck, and Luc Moreau. A Protocol for Recording Provenance in Service-oriented Grids. In *OPODIS*, 2004.
9. Paul Groth, Simon Miles, Weijian Fang, Sylvia C. Wong, Klaus-Peter Zauner, and Luc Moreau. Recording and Using Provenance in a Protein Compressibility Experiment. In *HPDC*, 2005.
10. Yi Huang, Alek Slominski, Chatura Herath, and Dennis Gannon. WS-Messenger: A Web Services based Messaging System for Service-Oriented Grid Computing. In *CCGrid*, 2006.
11. Gopi Kandaswamy, Liang Fang, Yi Huang, Satoshi Shirasuna, Suresh Marru, and Dennis Gannon. Building Web Services for Scientific Grid Applications. *IBM Journal of Research and Development*, 50(2/3):249–260, 2006.
12. James D. Myers, Carmen Pancerella, Carina Lansing, Karen L. Schuchardt, and Bret Didier. Multi-Scale Science: Supporting Emerging Practice with Semantically Derived Provenance. In *Semantic Web Technologies for Searching and Retrieving Scientific Data Workshop*, 2003.
13. Beth Plale. Resource Requirements Study for LEAD Storage Repository. Technical Report 001, Linked Environments for Atmospheric Discovery, 2005.
14. Beth Plale, Dennis Gannon, Dan Reed, Sara Graves, Kelvin Droegemeier, Bob Wilhelmson, and Mohan Ramamurthy. Towards Dynamically Adaptive Weather Analysis and Forecasting in LEAD. *LNCIS*, 3515:624–631, 2005.
15. Yogesh Simmhan, Beth Plale, and Dennis Gannon. A survey of data provenance in e-science. *SIGMOD Record*, 34(3):31–36, 2005.
16. Yogesh L. Simmhan, Beth Plale, and Dennis Gannon. A Framework for Collecting Provenance in Data-Centric Scientific Workflows. In *ICWS*, 2006.
17. Yogesh L. Simmhan, Beth Plale, and Dennis Gannon. Towards a Quality Model for Effective Data Selection in Collaboratories. In *IEEE Workshop on Scientific Workflows and Dataflows (SciFlow)*, 2006.
18. Jun Zhao, Carole Goble, and Robert Stevens. An Identity Crisis in The Life Sciences. In *IPAW*, 2006.
19. Yong Zhao, Michael Wilde, and Ian T. Foster. Applying the Virtual Data Provenance Model. In *IPAW*, 2006.

Exploring Provenance in a Distributed Job Execution System*

Christine F. Reilly and Jeffrey F. Naughton

University of Wisconsin–Madison
Department of Computer Sciences
1210 West Dayton Street, Madison, Wisconsin 53706, USA
{`chrissr`, `naughton`}@`cs.wisc.edu`

Abstract. We examine provenance in the context of a distributed job execution system. It is crucial to capture provenance information during the execution of a job in a distributed environment because often this information is lost once the job has finished. In this paper we discuss the type of information that is available within a distributed job execution system, how to capture such information, and what the burdens on the user and system are when such information is captured. We identify what we think is the key data that must be captured and discuss the collection of provenance in the Quill++ project of Condor. Our conclusion is that it is possible to capture important provenance information in a distributed job execution system with relatively little intrusion on the user or the system.

1 Introduction

Scientific computing applications are continuously growing in computational complexity and in the amount of data consumed and produced [1,2,3,4]. Many scientists utilize distributed job execution systems to meet their computational needs [5]. Within a distributed job execution environment much information is generated and exchanged regarding the execution and data access activities of the scientific application. This information can be used for tracking jobs through the system, recalling the activities of completed jobs, and for system accounting and debugging purposes. By archiving this information in a system that is visible to the user it can also be used to provide provenance information.

Our goal is to capture the provenance information that is available within a distributed job execution system. We specifically focus on the Condor system [5]; however, our discussion of the requirements for providing provenance in the context of a job execution system is applicable to the general category of distributed job execution systems. Condor is a distributed job execution system that runs on a dedicated cluster of machines, on idle desktop workstations, or on a combination of both environments [5,6]. This paper presents preliminary work on providing provenance information in Condor. In this work we explore

* To be published in: Proceedings of the International Provenance and Annotation Workshop, May 3-5, 2006, Chicago, IL. In: Lecture Notes in Computer Science.

categories of provenance in the context of a distributed job execution system, examine what provenance data is available in Condor, and discuss how this data can be captured.

The provenance gathered in Condor is an important part of the overall provenance of data items used by the jobs run in the system. This provenance must be gathered while a job is running because it is likely to be unavailable once the job has completed. Condor users have expressed the desire for being able to obtain provenance information about their jobs. Scientists are notorious for frequently changing their data and executable programs and keeping the same file name across multiple versions. As a result, it is often very difficult for a scientist to determine exactly which version of their program was applied to which version of their data to produce a given output. A second provenance need is the ability to determine if a job is affected by a hardware problem. A number of years ago, Intel reported a bug in the floating point unit of one of its processors. When this happened users wanted to know if their jobs were run on machines with a faulty processor. Our provenance system provides the information needed to meet both of these provenance needs.

We identify two types of provenance in Condor: logical provenance and infrastructure provenance. In our context, logical provenance consists of the input data items and executable program that create an output data item. Infrastructure provenance for an output data item consists of information about when the item was created and what parts of the Condor system were involved in the creation of the data item. These two types of provenance are described in detail in Sect. 2.

The first issue we discuss in this paper is what type and amount of information can be captured in Condor. It is widely agreed that provenance is useful for scientific applications [2,4] and that, intuitively, a system should provide as much information as possible. Two factors inhibit us from collecting all possible information: some types of information are difficult to detect, and it is infeasible to store all possible information. In Sect. 2 we examine the information available in Condor and discuss the benefits and drawbacks of various levels of provenance we are able to achieve with this information.

The second issue we address in this paper is how to gather provenance information in Condor. There are two entities that have information: the system and the user. Ideally we would like to design a provenance system that is transparent to the user and has few alterations to and impacts on Condor. Because both the user and the system hold provenance information, we require some amount of intrusion into each entity in order to gather provenance information. In Sect. 3 we discuss the provenance information gained from various levels of intrusion on both entities. We then discuss the provenance capabilities of the Condor Quill++ project in Sect. 4.

2 Categories of Provenance Information

This section examines the provenance information available in a distributed job execution system. In order to understand the space of provenance information we

divide it into various categories. The first division is on the type of information stored in the system: logical or infrastructure. Within each of these two types we present divisions of level of reproducibility and granularity.

Data provenance in a distributed job execution system involves three entities: the job execution system, the user, and the provenance system. The job execution system runs user submitted jobs that perform the transformation from input data to output data. The user submits jobs and manages data items and transformation functions. The provenance system stores information about data items, infrastructure items, and instances of data transformations.

2.1 Logical Provenance

Logical provenance describes the input data and transformation process that create some specific output data. This is the type of provenance that is discussed in much of the related work [4,7,8,9,10,11,12,13,14,15,16]. We define the logical provenance of an output data item as the input data items and transformation function that produced the output data item. Because we are looking at the portion of provenance that is related to a distributed job execution system, we focus only on how data is manipulated within the system. We assume the transformation function is deterministic and free of side-effects. Therefore, given the same input data the transformation function will always produce the same output data. The two variables we identify for logical provenance are its granularity and level of reproducibility. Granularity describes the level of detail represented by a data item. The level of reproducibility of logical provenance is determined by the method the system uses for identifying data items.

The level of granularity for logical provenance describes the level of detail represented by a data item. The desired granularity level depends on how the provenance information will be used [4]. Additionally, the granularity level that a system can provide depends on at what level that system can uniquely identify and track single data items. Some examples of granularities are file, portion of file, database tuple, and byte. We expect that in most distributed job execution systems a granularity of file level can be easily achieved because that is the granularity level at which these systems generally manage data.

The level of reproducibility provided by a logical provenance system is determined by what information is stored in the system for each provenance item. In this discussion we assume that every provenance item has a unique identifier that is provided by either the user or the job execution system. We define three reproducibility levels: inform, verify, and redo.

A system with logical inform provenance can tell the user what provenance item identifiers (e.g., file names) are associated with a specific use instance. If the user can associate the identifiers with the corresponding items in her possession then the user can reproduce the use instance. The system stores the unique name of the provenance item and identifies how it was used (i.e., input, executable, output). Logical verify provenance extends logical inform provenance by determining whether a proposed job is identical to a previous job, meaning that the two jobs have the same input and executable files. Verify provenance

is stronger than inform provenance because it detects, for example, if the same identifier is used for files that have different content. We suggest using a checksum to probabilistically verify that data items are identical because storing the entire data item is likely to require a large amount of storage.

A system with logical redo provenance is able to rerun a previously submitted job. This system stores the entire provenance item (e.g., entire data files and executables) along with its use type. Although logical redo provenance is an intuitively desirable feature [17], we do not view this level of reproducibility to be practical in most cases. Because redo provenance requires the system to store every data item, the storage requirements for such a system could quickly become unreasonable. One case where logical redo provenance may be practical is if the provenance system and user's data storage system are integrated such that the provenance system and user are using the same data storage system [18,19].

2.2 Infrastructure Provenance

Infrastructure provenance information describes the environment involved in the creation of a data item. There are two reasons why infrastructure provenance is useful. First, if the creation of a data item is dependent on specific environment variables then these variables are important portions of the provenance of the data item. Second, if part of the infrastructure is found to be defective then data items that were created using the defective infrastructure can be identified. Infrastructure provenance consists of the two same variables as logical provenance: granularity and level of reproducibility. However, these variables have slightly different definitions for infrastructure provenance.

For infrastructure provenance the level of granularity describes what information about the environment is stored by the provenance system. One category is information about the environment that created the data, such as the creation date, specific processor, operating system, and amount of memory. A second category is the system state when the data was created, for example the general system load, and the contents of the memory and disk on the machine that created the data.

For most systems there is a set of infrastructure information that is relatively easy to obtain and is fairly useful. Examples of such information are: creation time, specific processor, operating system, amount of memory, and general system load. If at a later date a processor, or the memory or disk associated with a specific processor, is found to be defective then the data items created with that processor can be identified. We can also picture infrastructure information that is difficult to record or recreate, such as the computer registry or specific state of the memory. Additionally some infrastructure information, such as the compiler used by the transformation function, is found at the user level. Depending on how provenance information is communicated to the provenance system this user level information may or may not be available.

Infrastructure provenance has two levels of reproducibility: inform and redo. For both levels the provenance system records infrastructure information specific

to a data transformation instance. A system with inform provenance can tell the user what infrastructure items were used in the creation of a specific data item. With redo provenance the system can recreate a specific data item using the same infrastructure as originally created that data item. We expect that in most cases infrastructure inform provenance is sufficient and redo provenance is unnecessary.

3 Obtaining Provenance Information

In this section we address the question of how the provenance system obtains provenance information. As in Sect. 2 we assume that three entities are involved in data provenance: the provenance system, the job execution system, and the user. The provenance system must obtain provenance information from a combination of both the user and the job execution system. We assume that at a minimum the job execution system provides the provenance system with system infrastructure information related to a job.

We describe the trade-offs between the amount of provenance information gathered and intrusions on the system and the user with a cube where the amount of intrusion on the system is on the x-axis, the amount of intrusion on the user is on the y-axis, and the amount of provenance information is on the z-axis (Fig. 1). The range of each axis is 0 to 1. A job execution system that has no provenance capabilities is located at the $(0,0,0)$ point. A system located anywhere on the back face of the cube, where the z-axis is equal to 1, collects all possible provenance information. The ultimate, and perhaps unachievable, goal is the $(0,0,1)$ point, where all provenance information is provided with no intrusion on either the system or user. Our goal is a system that provides a large amount of provenance information while having small intrusions on both the user and the job execution system. The point in Fig. 1 labeled “Goal” is intended to loosely suggest a desirable location, where the cost of moving further back in the cube would require dramatic increases in the intrusion on the user and/or system.

We discuss three configurations of how information is provided to the provenance system: job execution system based, user based, and shared. For each of these configurations we discuss the feasibility of implementing the method and the reliability of that method for gathering the provenance information. The feasibility of a configuration refers to how likely we think it is that current systems could and would be altered in order to implement the method. A high feasibility means that it is very likely that the configuration could be implemented because it requires few or no changes to current systems. A low feasibility means that it is unlikely that the configuration could be implemented because it requires many or difficult changes to current systems. The reliability of a configuration describes how likely we think it is that the method will capture provenance information. We have greater trust in the system than in the user for providing accurate provenance information. Therefore a high reliability means that provenance information is fully provided by the system and a low reliability means that provenance information is fully provided by the user.

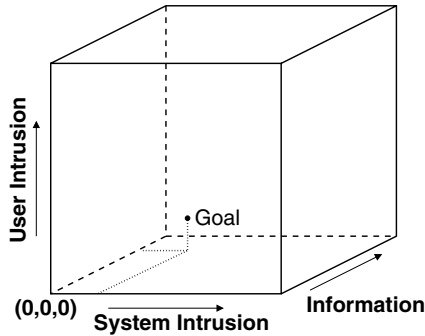


Fig. 1. Provenance Trade-offs Cube

When the job execution system provides all provenance information to the provenance system, the user can remain ignorant of the provenance system unless she requests provenance information. This scenario exists when the user intrusion equals zero on the provenance trade-off cube (Fig. 1). In such a scenario it is hard to capture all provenance information without intrusion on the system. For example, system intrusion is necessary for detecting access to files that are not declared in the job submission file. For such reasons we view a purely system-based approach to be of low feasibility.

If all provenance information is provided by the user then few to no alterations to the job execution system are necessary. This scenario exists when the system intrusion equals zero on the provenance trade-off cube (Fig. 1). We categorize this configuration of information gathering as high feasibility since few if any changes to the job execution system are necessary. However, this configuration has low reliability because we are completely depending on the user to provide accurate and complete information.

If both the user and the job execution system are aware of the provenance system then both can be relied upon for the gathering of provenance information. This is the scenario represented by the point labeled “Goal” in Fig. 1. In this case we rely on the job execution system to send messages to the provenance system. The user is required to be aware of the provenance system to enable the job execution system to send reliable messages to the provenance system. Exactly how the job execution system gathers provenance information and what the user must do depend on the structure of the job execution system.

4 Provenance in Condor

Quill++[20], an addition to Condor, was originally developed by the CondorDB team to provide better support for accounting and system monitoring, but we quickly realized that it could also be used to provide support for provenance. Quill++ writes information about machines, jobs, and workflows to logs then sniffs these logs and inserts the information into a central database. This ap-

proach allows Quill++ to make minimal changes to the Condor code and prevents Quill++ from blocking Condor. Quill++ has logical verify provenance capabilities and infrastructure inform provenance capabilities. Logical provenance information is stored at the granularity of files with the file identifier and checksum stored for each file. Infrastructure provenance includes information about machine hardware, software, and activity. Machine hardware information includes: processor identification, processor architecture, and amount of memory. Machine software information includes: operating system and Condor version. Machine activity information includes: if the machine is claimed by a Condor job, if the machine is idle, the time when Condor last heard from the machine, and statistics regarding machine load and activity.

Provenance information gathering is shared between the system and user. Information about jobs and machines is gathered from the Condor system by Quill++. File information is gathered from the job submission file. In order for Quill++ to obtain information about files the user must specify the file identifier and use type in the job submission file. It is possible that a job may use files that are not specified in the job submission file leaving Quill++ unaware of such files.

5 Conclusions and Future Work

We have shown that a good amount of provenance functionality can be achieved by storing information that is readily available within a distributed job execution system. For example, Quill++ stores information about the files used by a job, when and where the job ran, and some system state information. Quill++ imposes a minimal burden on the execution system and user, and provides what we hope is a useful amount of provenance information.

We have identified a number of items to explore in the future. Our first goal is to extend Quill++ to perform more system based gathering of provenance information by recording file information when Condor transfers files to the machine running a job. A second problem is to analyze the storage requirements of the provenance system in Quill++. Our preliminary analysis shows that in a cluster of thousands of machines the provenance portion of Quill++ generates a manageable amount of information over the period of one year. However, at some point in time the provenance information will need to be archived. Our third area of future work is to examine whether the provenance information regarding workflows must be explicitly recorded or if workflow provenance is recoverable from the provenance recorded for the component jobs.

Acknowledgments

This work was supported in part by National Science Foundation Award SCI-0515491.

References

1. Bose, R., Frew, J.: Lineage retrieval for scientific data processing: A survey. *ACM Computing Surveys* **37** (2005) 1–28
2. Jagadish, H., Olken, F.: Data management for the biosciences: Report of the NSF/NLM workshop on data management for molecular and cell biology, national library of medicine. Technical Report LBNL Report LBNL-52767, Lawrence Berkeley National Laboratory (2003)
3. Simmhan, Y.L., Plale, B., Gannon, D.: A survey of data provenance in e-science. *SIGMOD Record* **34** (2005) 31–36
4. Simmhan, Y.L., Plale, B., Gannon, D.: A survey of data provenance techniques. Technical Report IUB-CS-TR618, Computer Science Department, Indiana University, Bloomington, Indiana (2005)
5. Condor: Project homepage, <http://www.cs.wisc.edu/condor/> (2006)
6. Tannenbaum, T., Wright, D., Miller, K., Livny, M.: Condor – A distributed job scheduler. In Sterling, T., ed.: *Beowulf Cluster Computing with Linux*. MIT Press (2001)
7. Buneman, P., Khanna, S., Tan, W.C.: Why and where: A characterization of data provenance. In: *International Conference on Database Theory (ICDT)*. (2001)
8. Cui, Y., Widom, J.: Lineage tracing for general data warehouse transformations. In: *Proceedings of the 27th VLDB Conference, Roma, Italy*. (2001)
9. Cui, Y., Widom, J.: Lineage tracing for general data warehouse transformations. Technical report, Stanford University Database Group (2001)
10. Cui, Y., Widom, J.: Lineage tracing for general data warehouse transformations. *VLDB Journal* **12** (2003) 41–58
11. Fan, H., Poulouvasilis, A.: Tracing data lineage using schema transformation pathways. In B.Omelayenko, Klein, M., eds.: *Knowledge Transformation for the Semantic Web*. IOS Press (2003)
12. Foster, I., Vockler, J., Wilde, M., Zhao, Y.: Chimera: A virtual data system for representing, querying, and automating data derivation. In: *14th International Conference on Scientific and Statistical Database Management*. (2002)
13. Frew, J., Bose, R.: Earth system science workbench: A data management infrastructure for earth science products. In: *Thirteenth International Conference on Scientific and Statistical Database Management, Fairfax, Virginia*. (2001) 180–189
14. Widom, J.: Trio: A system for integrated management of data, accuracy, and lineage. In: *CIDR*. (2005)
15. Woodruff, A., Stonebraker, M.: Supporting fine-grained data lineage in a database visualization environment. In: *Proceedings of the 13th International Conference on Data Engineering, Birmingham, England*. (April 1997) 91–102
16. Cui, Y., Widom, J.: Storing auxiliary data for efficient maintenance and lineage tracing of complex views. In: *Proceedings of the International Workshop on Design and Management of Data Warehouses (DMDW), Stockholm, Sweden*. (2000)
17. Szomszor, M., Moreau, L.: Recording and reasoning over data provenance in web and grid services. In Meersman, R., Tari, Z., Schmidt, D.C., eds.: *On The Move to Meaningful Internet Systems 2003: CoopIS, DOA, and ODBASE - OTM Confederated International Conferences, CoopIS, DOA, and ODBASE 2003, Catania, Sicily, Italy, November 3-7, 2003*. Volume 2888 of *Lecture Notes in Computer Science*., Springer (2003) 603–620
18. Barga, R.: Automatic generation of workflow execution provenance. In: *International Provenance and Annotation Workshop (IPAW'06), Chicago, May 2006*. (2006) <http://www.ipaw.info/ipaw06>.

19. Braun, U., Garfinkel, S., Holland, D.A., Muniswamy-Reddy, K.K., Seltzer, M.I.: Issues in automatic provenance collection. In: International Provenance and Annotation Workshop (IPAW'06), Chicago, May 2006. (2006) <http://www.ipaw.info/ipaw06>.
20. Huang, J., Kini, A., Reilly, C., Robinson, E., Shankar, S., Shrinivas, L., DeWitt, D., Naughton, J.: An overview of Quill++: A passive operational data logging system for Condor. <https://www.cs.wisc.edu/condordb> (2006)

gLite Job Provenance^{*}

František Dvořák, Daniel Kouřil, Aleš Křenek, Luděk Matyska, Miloš Mulač,
Jan Pospíšil, Miroslav Ruda, Zdeněk Salvét, Jiří Sitera, and Michal Voců

CESNET z.s.p.o., Zikova 4, 160 00 Praha 6, Czech Republic
First.Last@cesnet.cz

Abstract. The Job Provenance (JP) service is designed to automate keeping track of computations on large scale Grids, giving thus users a tool to correctly archive information about their jobs and to re-submit any job in a reconstructed environment. JP provides a permanent minimal record of job (and its environment) related information, to which free-form user annotations can be added. JP also offers the capability of configuring any number of indexed logical views on the large collections of raw data, allowing efficient processing of even complex user queries selecting on both system data and the annotations. The scalable architecture, capable to handle millions of jobs in a single JP installation, and integrated into the EGEE gLite middleware environment is presented.

1 Job Provenance

New methods, instruments, and sensors are producing extreme amount of raw experimental data. The data must be further processed to provide a novel scientific insight and new knowledge. This leads to increased importance of processed (computed) scientific data whose amount growth at least exponentially. In this context the computation gets into the position of a traditional scientific experiment, including the principle that any results, should they be accepted by the community, *must be verifiable by re-doing the experiment*, i. e. re-running the computation. Consequently, an exact description of the computation — the *job* that produced a piece of data — contributes to the data provenance.

Unfortunately, many users, despite being scientists who are used to keep thorough track of their “real” experiments, tend to underestimate the importance of their computational experiments rather frequently. A common practice is running a job, analyzing and keeping the output, modifying some input parameters to run another iteration, but not archiving the original parameters. Then, after the user forgets the original parameters, the results of the first job get orphaned in the provenance sense, despite being archived otherwise.

In addition, the environment (e. g. versions of used software or internal parameters) of the job may affect the result of the computation, hence contributing to its provenance too.

We propose the Job Provenance service to automate the important but tedious task of keeping track of computations in the case of Grid jobs, as well as to allow effective access to the gathered data.

^{*} This work has been supported by the EU EGEE project INFISO-RI-508833.

1.1 Summarized Requirements

The designed service must keep a permanent (for several years) record of each registered job. This record contains information that is necessary to re-run the job, achieving the same results. The length of each record must be as minimal as possible, data which are stored elsewhere should not be included. The service should scale over the extent and throughput of current Grid middleware (e.g. EGEE reports [11] 20k jobs per day, i.e. 7.5M per year). The user should be allowed to add free-form annotations to each job record. A querying interface must be provided, allowing to inspect and retrieve records according to criteria specified on either the job data or the user annotations.

We have also more practically oriented goals, related to the fact that the service will be part of the gLite Grid middleware (Sect. 2.1). The first release should be deployed in at least moderate scale in 2006, so that feedback and eventual design revision may happen within the EGEE II project, i.e. by the end of 2007.

1.2 Related Work

A primary work on provenance is part of the EU Project *Enabling and Supporting Provenance in Grids for Complex Problems* [2]. They proposed a broad definition of provenance — *the provenance of a piece of data is the process that led to the data* — and presented proof of concept work on provenance in a Service Oriented Architecture [8]. Most current provenance architectures are concerned with incorporation of provenance capabilities into a Web Service-based Grid environments [4,5,10,12,13]. The usually considered scenario of execution of a composite service by trusted workflow engine provides the ability to collect and archive the provenance about the transformation of data during invocation of web services. This architecture is not practically suitable for our work as we need to support considerable number of “legacy”, non-WS-based services.

In EGEE related projects, number of HEP experiments have their own workflow systems that also provide some provenance and annotation (metadata) capabilities. The ATLAS experiment uses AMI database application framework [7] for production physics bookkeeping. AMI also manages “tasks” — the transformations that can be applied to datasets and their configuration parameters — and stores links between tasks and datasets. AliEn (ALICE Environment) [1] is a Grid framework that takes care of job splitting and execution, manages datasets, and keeps track of the basic provenance of each executed job or file transfer. AliEn File Catalog also provides interface for attaching new annotation (metadata) database tables to its standard directory tables. SAM [6] used in the DZero Experiment is a data handling system designed to store and retrieve files and associated metadata, including a complete record of the processing which has used the files. The major drawback of those systems is that they do not typically record activities of other middleware services and do not provide full provenance data due to their position at the top of the job submission (service invocation) chain. On the contrary, the Job Provenance is designed as an essential Grid service, defining a unified but still flexible framework for keeping records of jobs.

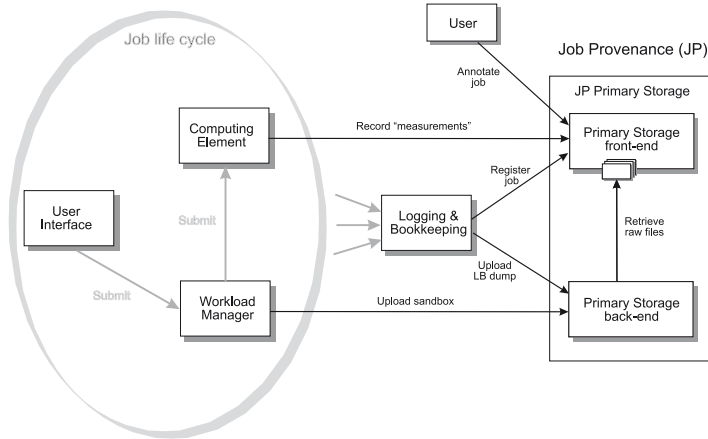


Fig. 1. Data flow into gLite Job Provenance

2 General Design

2.1 The Place of JP in gLite

The gLite Workload Management System (WMS) [3] consists of a set of Grid middleware services that facilitate convenient and efficient distribution and management of tasks across Grid resources. It accepts and dispatches users' requests for job management (mainly submission and cancellation), the decision which resource should be used is the result of a matchmaking process between requests and available resources. The gLite job properties and requirements are described in the job description language (JDL). The submission request can contain auxiliary input files (input sandbox) copied to an execution machine by WMS.

Currently, the information about jobs submitted to gLite Workload Management System is collected by the Logging and Bookkeeping (L&B) service. L&B tracks jobs in terms of events received from WMS components and CEs that are instrumented with the L&B calls. The events are collected by the bookkeeping server that processes them in a real time to give overall view on the actual job state. The user may query the bookkeeping server to obtain either the raw events or the computed job state, she may also register for receiving notifications on particular job state changes.

The L&B takes care of job information only during the job lifetime, purging the database when the job leaves the Grid. Also, L&B does not keep the input and output sandboxes, they are handled separately by WMS. The purpose of the gLite Job Provenance is to provide the permanent storage of the job related information as stored within the L&B, to couple it with the input sandboxes and other system oriented information necessary to reproduce the environment where a particular job run. Fig. 1 depicts basic gLite middleware components and their interaction with the Job Provenance.

2.2 Data Gathered from Middleware and User Annotations

The storage capacity requirement for a service to keep permanent track of a very large number of Grid jobs may become enormous. Consequently the data recorded for each job must be strictly limited. As a rule of thumb we store only volatile data which are neither stored reliably elsewhere nor are reproducible by the job. The data gathered from the gLite middleware fall into the following categories:

- job inputs, directly required for job re-running
 - complete job description (JDL) as submitted to WMS
 - miscellaneous input files (gLite WMS input sandbox) provided by the user (but job input files from remote storage *are not* copied to JP)
- job execution track, witnessing the environment of job execution
 - complete L&B data, i. e. when and where the job was planned and executed, how many times and for what reasons it was resubmitted etc.
 - “measurements” on computing elements, e. g. versions of installed software, environment settings etc.

In addition, the service allows the user to add arbitrary annotations to a job in the form of “name = value” pairs. Annotations can be recorded either during the job execution or at any time afterward. Besides providing information on the job (e. g. it was a production-phase job of particular experiment) these annotations may carry information on relationships between the job and other entities like external datasets, forming the desired data provenance record.

2.3 Raw and Logical Data Representation

At the *raw level*, data enter JP and are stored in two ways: (i) small size *tags*, i. e. “name = value” pairs, and (ii) uploaded bulk files. At the *logical level*, any piece of information stored in JP is represented as a value of particular named *attribute*. In this representation both the user annotations and the “system” middleware data are unified in a single view.

Data stored as tags map to attributes in a straightforward way, name and value of the tag becoming name and value of an attribute. An uploaded file (L&B log, job sandbox, ...) is usually a source of multiple attributes. JP defines a *file-type plugin interface API*; the task of the plugin is parsing a particular file type and providing calls to retrieve attribute values.

For the purpose of extensibility an attribute name always falls into a namespace. Currently we declare namespaces for JP system attributes (e. g. job owner or registration time), attributes inherited from L&B, and unqualified user tags.

2.4 Typical Usage

Propagation of data from other middleware components to JP is done transparently. The user may specify both the destination JP and which data are gathered via special parameters in the job description, however, these settings

may be overridden by WMS or CE policy. The user also interacts with JP directly when recording annotations. Retrieval of information on *a concrete job* is fairly straightforward—the job is the primary entity in JP and all data are organized on a per-job basis, hence easily available.

But the principal purpose of JP is *searching for jobs* according to some criteria, notably jobs that either produced or used a given piece of data, freeing the user of the burden to keep complete records on her jobs. Such a search would result in scanning through all data stored in JP which are expected to be huge, being unacceptable for frequent user queries. Instead we define an architecture that allows batch pre-processing of configurable queries. The result of such query, a superset of certain user query type, is further indexed in order to provide fast response to concrete user queries.

3 Architecture

JP is formed of two classes of services: permanent *Primary Storage* accepts and stores job data while possibly volatile and configurable *Index Servers* provide an optimized querying and data-mining interface to the end-users.

3.1 Primary Storage

The JP *Primary Storage* (JPPS) is a permanent service responsible for gathering the job data and their long-term archival. The primary data are kept in as compact form as possible, and only minimal metadata (job ID and owner, registration time) are maintained and indexed.

A single instance of JPPS is formed by a front-end, exposing its operations via a web-service interface¹, and a back-end, responsible for actual data storage and providing the bulk file transfer interface. In the current implementation metadata are stored in a relational database. The back-end uses Globus grid-ftp server enriched with authorization callbacks accessing the same database to check whether a user is allowed to upload or retrieve the given file. Both the front- and back-ends share a filesystem so that the file-type plugins linked into the front-end access their files via POSIX I/O.

Job registration. Each job has to be explicitly registered with JP. The registration is done transparently, the L&B server calls JPPS front-end `RegisterJob` operation upon job submission (in parallel with the job registration in L&B).

Data upload. The `RecordTag` operation records the “name = value” tags.

Uploading a bulk file is a more complex, three-stage sequence:

- Call `StartUpload` front-end operation. If authorization check succeeds, the service responds with upload URL with a limited time span.
- Upload the file to the specified URL. The authorization callback of the grid-ftp server checks whether this particular URL was “opened” in the previous step for the concrete user and is still valid.

¹ Described in detail in [9], documented web service definitions can be found at <http://egee.cesnet.cz/en/WSDL/>

- Confirm the finished upload with `CommitUpload` front-end operation. This “closes” the URL for upload, makes the file available to the front-end, and opens the URL for eventual download.

Index Server feed is the data-mining interface called by JP Index Servers described in Sect. 3.2.

Data retrieval. The only direct data retrieval supported by JPPS is keyed by ID of jobs. There are two operations, both taking job ID as their argument:

- `GetJobAttributes` retrieves attribute values, either stored as user tags or extracted from uploaded files via the file-type specific plugins.
- `GetJobFiles` returns URL’s pointing to the job files stored at the Primary Storage back-end. The user may retrieve the raw files via the back-end interface, and parse them on her own.

Only limited number of JPPS installations must be deployed even on a large Grid to concentrate the provenance data. At most one JPPS per a virtual organization is envisaged for the EGEE environment. This mean each JPPS must be able to deal with data on millions of jobs. The typical size of an L&B dump is around 10kB per compressed record, and gLite users are encouraged not to use large job sandboxes, too. Consequently, the back-end storage requirements are at the order of 10-100 GB. JPPS metadata are formed by a single tuple for each job and for each file, with unique indices on job ID and file name. The used MySQL database engine is capable to handle millions of such records.

Primary Storage covers the first set of requirements specified in Sect. 1.1 — storing a compact job record, allowing the user to add annotations, and providing elementary access to the data.

3.2 Index Server

The role of *Index Servers* (JPIS) is processing and re-arranging the data from Primary Storage(s) into a form suitable for frequent and complex user queries. A typical interaction is shown in Fig. 2.

1. The user queries one or more JPIS, receiving a list of ID’s of matching jobs.
2. JPPS is directly queried for additional job attributes or URL’s of stored files.
3. The required files are retrieved.

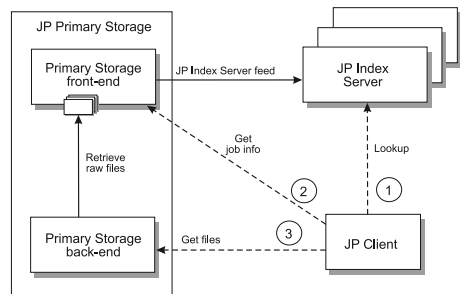


Fig. 2. Index Server interactions

The current format of the user query is a list of lists of conditions. A condition is comparison (less, greater, equal) of an attribute w. r. t. a constant. Items of an inner list must refer to the same attribute and they are logically or-ed. Finally the inner lists are logically and-ed. According to our experience with the L&B

service, this query language is powerful enough to satisfy user needs while simple enough to allow efficient implementation.

Index Servers are created, configured, and populated semi-dynamically according to particular user community needs. The configuration is formed by:

- one or more Primary Storages to contact,
- conditions on jobs that should be retrieved,
- list of attributes to be retrieved,
- list of attributes to be indexed — a user query must refer to at least one of these for performance reasons.

The set of attributes and the conditions specify the set of data that is retrieved from JPPS, and it reflects the assumed pattern of user queries. The amount of data fed into a single JPIS instance is assumed to be only a fraction of data in JPPS, both regarding the number of jobs, and the number of distinct attributes.

Communication between JPIS and JPPS involves two complementary web-service operations: JPIS calls the **FeedIndex** operation of JPPS, specifying the list of attributes and conditions. Unlike the user queries, the query on JPPS is a single and-ed list, allowing less complex processing on JPPS where significantly larger data set are involved. JPPS responds by calling the **UpdateJobs** operation of JPIS (repeatedly to split up large dataset).

The following flags in the **FeedIndex** call specify the query mode:

- *history* — JPPS should process all its stored data, giving the user the guaranty that if her query is a logical restriction of the JPIS configuration, it returns a complete result. This type of query is usually necessary to populate JPIS but it imposes rather high load on JPPS.
- *continuous* — JPIS registers with JPPS for receiving *future updates* when data matching the query arrive. This type of query allows JPIS to be kept up to date while imposing minimal load on JPPS.

The current JPIS implementation keeps the data also in a MySQL database. Its schema is flexible, reflecting the Server configuration (columns are created to hold particular attribute value, as well as indices). There is no prescribed relationship between Primary Storage and Index Server installations. An Index Server may retrieve data from multiple Primary Storages and vice versa.

4 Conclusion

The gLite Job Provenance service is a specific provenance that helps to keep track of millions of jobs, their environments and inputs in large Grids. Data collected directly by the Grid middleware could be arbitrarily annotated at any time. While designed as an independent service, the JP implementation is integrated into the EGEE gLite middleware.

End users can query JP to obtain data about a specific job. In addition, user communities are supposed to install JP Index Servers to process continuously the JP data (even collecting data from several JP Primary Storages) and to

provide logical views optimized for specific users' queries. In this way the users obtain more complex information and knowledge from the JP data. Independent Index Servers can be deployed in order to provide different logical views, giving thus fast access to even the largest collections of job related data.

The gLite Job Provenance service is currently being deployed on the EGEE Grid. A set of Index Servers is pre-configured to provide most common logical views on the data. The large scale deployment will test the architecture and its scalability and it will also provide a necessary feedback for further extensions: the plug-ins, specific IS configurations, expressing power of the annotations, specific tools for manipulation with the primary data (e.g. automatic and parametrized job re-submission), etc. Also, the deployment will be used to test and extend the current simple authorization model used in the first JP implementation.

References

1. AliEn (ALICE ENvironment). <http://aliceinfo.cern.ch/AliEn>.
2. EU FP6 Programme Enabling and Supporting Provenance in Grids for Complex Problems. <http://twiki.gridprovenance.org/bin/view/Provenance/ProjectInformation>.
3. gLite—Lighthouse Middleware for Grid Computing. <http://glite.web.cern.ch/glite/default.asp>.
4. Mygrid Provenance Outline. <http://phoebus.cs.man.ac.uk/twiki/bin/view/Mygrid/ProvenanceOutline>.
5. PASOA: Provenance Aware Service Oriented Architecture. <http://twiki.pasoa.ecs.soton.ac.uk/bin/view/PASOA/AboutPasoa>.
6. SAM (Sequential data Access via Meta-data). <http://d0db.fnal.gov/sam/>.
7. The Atlas Metadata Interface. <https://atlastagcollector.in2p3.fr:8443/AMI/>.
8. Liming Chen, Victor Tan, Fenglian Xu, Alexis Biller, Paul Groth, Simon Miles, John Ibbotson, Michael Luck, and Luc Moreau. A proof of concept: Provenance in a Service Oriented Architecture. In *Proceedings of the fourth UK e-Science All Hands Meeting, Nottingham, UK*, 2005.
9. EGEE JRA1. EGEE Middleware Design—Release 1. <https://edms.cern.ch/document/487871/>.
10. Paul Groth, Michael Luck, and Luc Moreau. A protocol for recording provenance in service-oriented grids. In *Proceedings of the 8th International Conference on Principles of Distributed Systems (OPODIS04)*, 2004.
11. EGEE JRA2. Job Metrics. <http://egee-jra2.web.cern.ch/EGEE-JRA2/QoS/JobMetrics/JobMetrics.htm>.
12. Shrija Rajbhandari and David W. Walker. Support for Provenance in a Service-based Computing Grid. In *Proceedings of the third UK e-Science All Hands Meeting, Nottingham, UK*, 2004.
13. Paul Townend, Paul Groth, and Jie Xu. A provenance-aware weighted fault tolerance scheme for service-based applications. In *Proceedings of the 8th IEEE International Symposium on Object-oriented Real-time distributed Computing (ISORC 2005)*, 2005.

An Identity Crisis in the Life Sciences

Jun Zhao, Carole Goble, and Robert Stevens

School of Computer Science, University of Manchester, M13 9PL, U.K.
{zhaoj, carole, robert.stevens}@cs.man.ac.uk

Abstract. ^{my}Grid is an e-Science project assisting life scientists to build workflows that gather data from distributed, autonomous, replicated and heterogeneous resources. The provenance logs of workflow executions are recorded as RDF graphs. The log of one workflow run is used to trace the history of its execution process. However, by aggregating provenance logs of many workflow runs, one may gather the provenance of a common data product shared in multiple derivation paths. A successful aggregation relies on accurate and universal identification of each data product. The nature of bioinformatics data and services, however, makes this difficult. We describe the identity problem in bioinformatics data, and present a protocol for managing identity co-references and allocating identity to gathered and computed data products. The ability to overcome this problem means that the provenance of workflows in bioinformatics and other domains can be exploited to enhance the practice of e-Science.

1 Introduction

^{my}Grid ¹ is an e-Science project providing middleware services to assist bioinformaticians to perform *in silico* experiments [1]. ^{my}Grid uses workflows to orchestrate, access and interoperate a large number of public databases and applications, and manage those experiments and their *outcomes*, including data products, their provenance and experiment conclusions, using semantic-based metadata and data technologies [2]. Taverna, the workflow environment and workbench in ^{my}Grid, enables scientists to design and execute workflows, providing access to over 3,000 bio-resources. These services are mixtures of web services, grid services, java applications, database queries and scripts. Taverna has been used for gene alerting, gene and protein sequence annotation, proteomics, functional genomics, chemoinformatics, systems biology and protein structure prediction applications. Workflows have been used to identify a mutation associated with the autoimmune disorder Graves' Disease in the I kappa B-epsilon gene [3] and build the first complete and accurate map of the region of chromosome 7 involved in Williams-Beuren Syndrome (WBS) [4].

As part of the experiment design, scientists construct executable workflow definitions, written in Taverna's Scuf language [4], binding specific data, parameter settings and the end points of the services to be executed. Each execution of a workflow definition becomes a workflow run. Collections of workflow definitions

¹ <http://www.mygrid.org.uk>

and workflow runs contribute to an overall experiment. myGrid collects both data produced during workflow runs and the provenance of these data products [2]. The provenance log of one workflow run is used to trace: the history of execution process, (e.g. services used); the origin of a data product, (e.g. the database or intermediate data product); and the ownership and intellectual property of each run (e.g. who and when). Ownership, purpose, etc are provenance annotations over experimental collections of workflow definitions as well as an individual workflow run. The provenance of each data product, gathered or computed, is automatically captured by detecting events during workflow enactment to form a dependency graph. Annotations of ownership and purpose are manual assertions of fact or opinion by the scientist on the data, processes or sub-graphs.

The workflows developed in the life sciences have a number of properties that impact on our provenance collection:

- The workflows are largely data pipelines, frequently generating data collections and then iterating in turn over each item in the collection.
- A workflow data product can be (a) a newly generated original data object, or (b) a pre-existing data object gathered from an external resource. Thus, when we refer to data products, these can be pre-existing objects that have been retrieved from an external collection (*gathered* data) or freshly computed data values (*computed* data).
- As public resources regularly change their content, the same workflow is rerun repeatedly over the same resources, perhaps with different parameter settings. The data products acquired by different runs are compared, merged and aggregated. A workflow can thus be executed repeatedly by the same user at a different time or location, or by different users from different research groups or institutions.
- The same data product may be acquired by different workflow runs under different experimental contexts, e.g. the user, the workflow definition, or the time of a run etc.

The final two points are important. Bioinformatics is an exploratory scientific discipline. Varying the settings of repeated executions might lead to completely different outcomes. Results and their provenance from repeated executions need to be accumulated to verify an ultimate conclusion. Thus, scientists need to put the provenance records to use: to aggregate, integrate and compare the provenance records for a common data product produced by multiple workflow runs.

Definition 1. Consider a data product d acquired in a workflow run r ,

- the provenance log of r forms a graph $P(r)$;
- the provenance log of d in r is a subgraph of $P(r)$: $P(d, r)$.

Then, for d acquired in both r_1 and r_2 :

- Provenance *aggregation* for d is to gather the provenance graphs $P(d, r_1)$ and $P(d, r_2)$.

- Provenance *integration* for d is the merging of the aggregated provenance graphs: $P(d, r_1) \cup P(d, r_2)$ [5].
- Provenance pair-wise *comparison* for d from runs r_1 and r_2 is the computed difference between the two provenance graphs: $P(d, r_1) - P(d, r_2)$ [6].

To aggregate, integrate and compare $P(d, r_i)$ ($i = 1, 2, \dots, n$) for d requires two things:

1. **a mechanism to merge and differ provenance graphs:** ^{my}Grid represents the workflow provenance metadata using technologies drawn from the semantic web community, chiefly the Resource Description Framework (RDF) ². RDF is essentially a simple graph data model. The RDF data store and query languages provide mechanisms for graph fusion and graph manipulation as well as querying.
2. **a mechanism to manage data object identity:** When we merge and differ provenance graphs, it is helpful if the same data object - a protein sequence, a gene, a database entry - has the same identity regardless of its origin. RDF provides an explicit identification system, Universal Resource Identifiers (URIs), for identifying resources to allow metadata about a resource to be merged from several sources. We identify gathered data objects, computed data products, workflows, parameters, etc by allocating LSIDs (Life Science Identifiers) [7] to them.

Representing provenance using RDF and LSIDs enables us to potentially aggregate multiple provenance graphs, $P(d, r_i)$ ($i = 1, 2, \dots, n$). Although the RDF-based graph model and associated manipulation and query mechanisms lend themselves to the support of cross-run or cross-workflow provenance aggregation and integration, the allocation and management of identity is problematic. LSIDs are proposed as global unique identifiers (GUIDs) by the life science community [7,8]. This scheme has been applied to major life science databases, such as NCBI ³, UniProt ⁴, and Affymetrix ⁵. In many e-Science domains, such as chemistry, physics, astronomy, etc, GUIDs for data objects are taken for granted. An example is the Digital Object Identifier (DOI) [9] for digital publications.

However, multiple identities are allocated for the same data object in life sciences. The identity allocation scheme in Taverna does not guarantee a universal identity to be allocated for *equivalent* data (defined in Section 2) produced in multiple runs. These multiple identities for the same or equivalent data are called *polyonomous* identities ⁶, which lead to an identity crisis in inter-run provenance aggregation, integration and comparison.

The rest of this paper is organized as follows: Section 2 describes the motivation for managing data identities and presents a simple, real workflow. Section 3 highlights the life science identity landscape, showing how the difficulties in

² <http://www.w3.org/RDF/>

³ <http://www.ncbi.nlm.nih.gov/Genbank/>

⁴ <http://www.ebi.uniprot.org/>

⁵ <http://lsid.biopathways.org/authorities.shtml>

⁶ <http://dictionary.reference.com/search?q=Polyonomous>

allocating identities to data products in a coherent fashion have consequences on the recording and aggregating of provenance records. In Section 4, we propose a new identity protocol to construct identity co-references and the introduction of a new identity naming scheme. We show how identity co-references can help when comparing provenance generated in repeated runs of the presented real workflow. Related identity work and identity management in provenance are described in Section 5. We conclude with a discussion and summary of the characteristics of the identity problem.

2 Collecting Provenance from a Taverna Workflow

Figure 1 schematically presents a simple workflow (WF1) from the WBS study. This workflow identifies a collection of DNA sequences from a database, similar to the query sequence. Step 1 invokes the BLAST (Basic Local Alignment Search Tool) service [10] using the initial query sequence and a set of configuration parameters. BLAST detects regions of similarity embedded in otherwise unrelated proteins or nucleic acids. Step 2 simplifies the BLAST report data product and extracts the DNA sequences contained from this report. Step 3 retrieves the GenBank report for each DNA sequence produced in step 2 and produces a collection of GenBank reports.

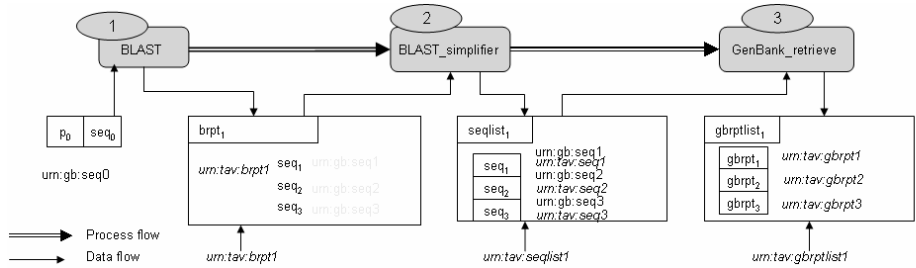


Fig. 1. Workflow (WF1) which forms part of the WBS case study. The BLAST report $brpt_1$, the sequence list $seqlist_1$, the GenBank report list $gbrptlist_1$ and each GenBank report $gbrpt_i$ are computed data products. Each sequence (seq_i) pre-exists, gathered and processed by the workflow. During the execution of the workflow, each seq_i has an external identity (e.g. $urn:gb:seq1$). This identity is lost as it becomes a text string by step 1, which is subsequently recovered by step 2 and associated with an additional Taverna identity.

The bottom of the figure shows the data products consumed and produced in one run of WF1.

- In step 1, the input to the BLAST service is a DNA sequence (seq_0) to align against, and a set of parameter settings p_0 for invoking the service, which includes: the *database* $_0$ of sequences, the statistical significance threshold *value* $_0$ for reporting sequence matches, and the maximum number of reported *scores* $_0$ in the BLAST report. The output of step 1 is a BLAST

report, $brpt_1$, containing a collection of sequence data entries retrieved from $database_0$: $Contains(brpt_1) = \{seq_i, 1 \leq i \leq n\}$. The sequences and their identifiers are embedded as text in the report, along with other materials.

- In step 2, the $brpt_1$ produced in step 1 is parsed and simplified by the **BLAST_Simplifier** service. This service extracts the sequence entries in $brpt_1$ to recover a collection of DNA sequence data objects: $seqlist_1 = \{seq_i, 1 \leq i \leq n\}$.
- In step 3, a GenBank report $gbrpt_i$ is retrieved by the **GenBank_Retrieve** service for each sequence in $seqlist_1$ produced in step 2. The output of step 3 is a collection of GenBank reports: $gbrptlist_1 = \{gbrpt_i, 1 \leq i \leq n\}$.

2.1 Gathered and Computed Data

A data product can be either *computed* or *gathered*, which decides the identities it might be associated with. A data product can be either *atomic* or a *collection*, which decides the provenance metadata captured for it in ^{my}Grid.

- A computed data product is generated as a consequence of a workflow execution. The BLAST report $brpt_1$, the GenBank report $gbrpt_1$, the collection of sequences $seqlist_1$ and the collection of GenBank reports $gbrptlist_1$ are computed data products.
- A gathered data product is one that was pre-existing and has been retrieved from external databases. Each protein sequence entry seq_i from the sequence database, contained in the $brpt_1$ is a gathered data product.

A computed or gathered data product can be either an atomic data product or a collection data product. Each collection data product contains a collection of elements, which are either atomic or collection data products, following the usual recursive composite pattern. Whether a data product is atomic or a collection can be rather subtle, dependent on the domain view or the transportation view:

- A domain collection is classified by its data content, for example, $brpt_1$ is collection data product at the domain level, containing a collection of gathered sequences $\{seq_i\}$. Every seq_i is an atomic data product.
- A transportation collection is decided by whether the data product is treated as a single data product or as a list of data products in Taverna. For example, $brpt_1$ is an atomic data product at the transport level when it is transferred between services in the workflow runs. The $seqlist_1$ and the $gbrptlist_1$ in Figure 1 are collection data products, as they are transferred as *lists* between services.

2.2 Using Provenance Graphs

In ^{my}Grid, data provenance gathered in each workflow run forms a graph, shown in Figure 2, with data products as the nodes and their *provenance* relationships as the edges. Consider a data product d_i produced in a run, either an atomic or a collection, there are two types of provenance relationships recorded in the ^{my}Grid provenance:

- *derivedFrom*: d_i is derived from another data product d_j , or d_i is derived from a set of parameter settings p_j . For example, the BLAST report $brpt_1$ is derived from the input sequence seq_0 and from the set of parameters $p_0 = \{database_0, evaluate_0, score_0\}$.
- *elementOf*: d_i could be an element of a collection data product d_j ($d_i \neq d_j$). For a seq_i of $seqlist_1$, atomic seq_i is an element of the collection data product $seqlist_1$. The ^{my}Grid provenance model only captures the relationship between a transportation collection and its elements.

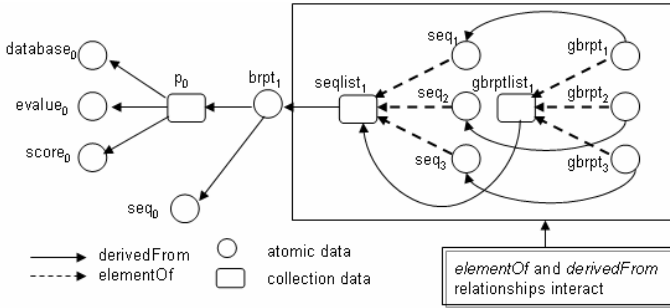


Fig. 2. Data provenance graph formed in one run of WF1

A bioinformatics workflow is often repeated over the same resources or varying settings. For example, considering the WF1, one might use:

1. the same BLAST service and same parameter settings to obtain updated data products. As the public databases are frequently updated, WF1 needs to be frequently repeated in order to collect updated sequences.
2. the same BLAST service but different parameter settings to look for the best parameter settings for this workflow.
3. different BLAST services, such as DDBJ BLAST⁷, WU-BLAST⁸, to look for the best service for this workflow or to verify that consistent results can be obtained with varying BLAST services.

The data products of these repeated runs of WF1 are compared to reach conclusions, e.g. an updated sequence was produced in a rerun of WF1, or “score=100” is the best parameter setting for running WF1. These conclusions need to be justified using the provenance of experiment data products.

In repeated workflow runs, multiple data provenance graphs are produced, which contain some similarities and differences. The same data object can be gathered in different runs; e.g. the protein sequence seq_1 appears as a result in both r_1 and r_2 of WF1. Thus both provenance graphs $P(seq_1, r_1)$ and $P(seq_1, r_2)$ have seq_1 in *common*. Data objects that contain the same data values can be

⁷ www.xml.nig.ac.jp/wsd1/index.jsp

⁸ <http://blast.wustl.edu/>

computed in different runs, e.g. $brpt_1$ from r_1 and $brpt_2$ from r_2 contain exactly the same sequences, despite changes in time, parameters or the contents of the external database. Thus $brpt_1$ and $brpt_2$ in the two graphs $P(brpt_1, r_1)$ and $P(brpt_2, r_2)$ correspond. In this paper *equivalent* data products include both gathered products in common and corresponding products that are computed.

In order to *aggregate* the provenance of a d computed or gathered in multiple runs, we need to identify this d produced in each run. In order to *integrate* and *compare* the multiple provenance graphs of d , i.e. $P(d, r_i)$ ($i = 1, 2, \dots, n$), we need to identify all the equivalent data products and parameter settings recorded in these graphs. The provenance graph in Figure 2 is represented by RDF in ^{my}Grid. Each data product and control parameter in this graph is identified by a URI. We need to manage the identities to make sure that equivalent data products and parameters are identified uniquely and universally across workflow runs. In the next section we explain the identity issues for WF1.

3 Identities

In Figure 1, all identities of all data products by a workflow are managed by the LSID protocol. An LSID consists of five parts separated by colons: a prefix (`urn:lsid`); the authority name (`www.mygrid.org.uk`); the authority-specific data namespace (`data`); the namespace-specific object identifier (`49841`) and a version number of the object (`1`) leading to a URI: `urn:lsid:www.mygrid.org.uk:data:49841:1`. An LSID authority for a resource allocates an identity and resolves it for the resources for which it is an authority (and no others), guaranteeing that the data is immutable. Each data provider has a responsibility for managing its own LSID authority. Even when resources are replicated locally a different, local LSID authority is in place.

A *gathered* data product may carry an external identity, but a *computed* data product is assigned with a Taverna identity. This impacts on the protocol for managing data identities as shown in Section 4. Using the workflow WF1 in Figure 1, we now explore external identity allocation by data resources and internal identity allocation by Taverna.

3.1 External Identity: Resource Generated Identities

At least 700 different, heterogeneous resources are available in life sciences [11]. The autonomy of data providers enables rapid generation and deployment of new resources, but they rarely conform to any community-wide standards, often allocating different identities for a common data object. We find the following situations:

- equivalent data objects in different databases. For example, “`gi:15145617`” (GenBank) and “`ac073846`” (EMBL-Bank⁹) are the same DNA sequence. The protein “Dual specificity DE phosphatase Cdc25C” is identified as “`aaa35666`” in GenBank and “`p30307`” in UniProt.

⁹ <http://www.ebi.ac.uk/embl/>

- equivalent data objects in different replicas, a variation of the previous point. A resource is often replicated remotely or locally, and sometimes locally extended or customised. For example, the same sequence is “`urn:lsid:myg:ac073846`” for a local copy of EMBL-Bank in ^{my}Grid with its own LSID authority.
- equivalent data objects from different workflows. The invocation of a Kyoto Encyclopedia of Genes and Genomes (KEGG¹⁰) pathway service (instead of a BLAST service) produces a pathway result containing a collection of protein sequences. Some of the sequences in the *brpt*₁ in Figure 1, also appear in the pathway data product, but now with KEGG LSIDs.

These arrangements result in polyonomous external identities for equivalent data products gathered from different databases, replicas or by different services.

3.2 Taverna Identity: Workflow Generated Identities

Figure 3 gives the ^{my}Grid LSID allocation architecture. In a workflow run, when a data product is passed to the enactor, it is allocated a Taverna LSID by the Taverna LSID authority. This identity is associated with the data product when it is stored or passed to invoked other services by the enactor. The data products acquired (gathered or computed) by a run are stored in a customized database as part of the workflow or a local “catch all” store, the relational data store BACLAVA. This identity is also used to store the RDF provenance metadata of this data product in the metadata store, KAVE. Data and provenance are immutable in ^{my}Grid. Once data products and provenance are preserved, they should not be deleted or altered. Provenance of a data product can be augmented with more provenance metadata. A client communicates the ^{my}Grid data and provenance repositories over the network by the LSID protocol [8] to retrieve data or metadata of a data product by its LSID.

Migration Polyonomy. Data products generated by Taverna and stored in a database or the BACLAVA store can be archived or replicated among scientists in their own file stores. A migration polyonomy is caused by the failure to migrate data identities with the data when the data are curated. For instance, when a data product is copied from the BACLAVA store to a personal file system, its identity is deprecated and replaced with a new identity such as a path to access the file system.

Execution Polyonomy. In each independent run, the Taverna LSID authority is ignorant of the existence of common or corresponding data products produced in different runs and the existence of the polyonomous identities allocated for these data products. Therefore, polyonomous identities for common or corresponding data products are produced by repeated executions:

1. *corresponding computed data products.* Two corresponding *brpt*₁ and *brpt*₂ generated in two runs of WF1 contain exactly the same protein sequence

¹⁰ <http://www.genome.jp/kegg/>

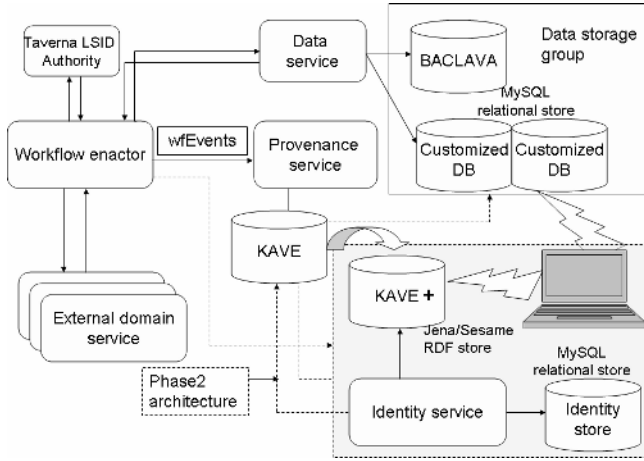


Fig. 3. ^{my}Grid’s architecture for allocating LSIDs for data products during workflow executions. The dashed part represents the extended architecture when introducing the identity service to manage the identities for common or corresponding data products.

objects. $brpt_1$ and $brpt_2$ are identified by different Taverna identities as they are from different runs and are, in fact, different reports.

2. *common gathered data products.* Let d_1 be the same seq_1 gathered in two runs r_1 and r_2 , two provenance graphs $P(d_1, r_1)$ and $P(d_1, r_2)$ share this common d_1 (see Figure 4(a)). This d_1 should be given the same identifier in order to assert that d_1 in r_1 and r_2 are equivalent. However, the processing of the workflows means that this is not the case.

- In Figure 1, when step 1 is finished, the computed data product $brpt_1$ is stored and allocated a Taverna LSID `urn:tav:b1`. The gathered data objects, i.e. the $\{seq_i\}$ contained within $brpt_1$ are neither extracted nor allocated with any LSIDs. This is because they have been turned into text by BLAST and their external identities, e.g. `urn:gb:seq1`, are contained in their data contents as strings.
- When step 2 is finished, each sequence, such as seq_1 , has been extracted and stored. Because it is a new object, recovered by post-processing, it is automatically allocated a Taverna LSID `urn:tav:seq1`, despite the fact it already has an external LSID that it carried. In each workflow run of WF1, if seq_1 appears it is given a new, different Taverna LSID. Thus the same data product has its external LSID and, for each run, a Taverna LSID.

Two cases further compound the problem:

1. corresponding computed collection data products at the transportation level. For instance, the data product from the `GenBank_retrieve` service $gbrptlist_1$ contains a collection of GenBank reports $\{gbrpt_i\}$. This collection data prod-

uct and its elements are always allocated new, different Taverna identities each time they are produced.

2. equivalent nested data objects in a data product. For instance, each sequence seq_i contains some nested data objects such as the species data object. These nested data objects are allocated with new, different Taverna identities each time they are extracted.

Polygonomies are not only due to inadequate attention to our identity allocation mechanism, but are inevitable. Firstly, we need to differentiate the data product and its data derivation path produced in one context with its equivalent data product produced in another context. Figure 4(a) shows the two derivation paths for the data product d_1 that were gathered in different runs. If during the provenance collection this d_1 was identified by the same identity, as shown in Figure 4(b), only the merged derivation path will be kept in KAVE. It becomes difficult to retrieve "the data product which d_1 was derived from that was produced in run r_1 ". myGrid tries to solve this problem by incorporating more context information with a data product using named graphs [12], as discussed in Section 5.

Secondly there are potential computation costs of avoiding publishing polygonomous identities at run time for equivalent data products. The equivalence between computed data products is based on their values most of the time, which is inefficient. For instance, the identity correspondence of our example BLAST reports cannot be decided based on the identities of the sequence data objects contained in these data products, as polygonomous identities are published by different databases for the same sequence. Evaluating the equivalence of data products at run time could slow down the workflow enactor, but is achievable by a post workflow enactment process.

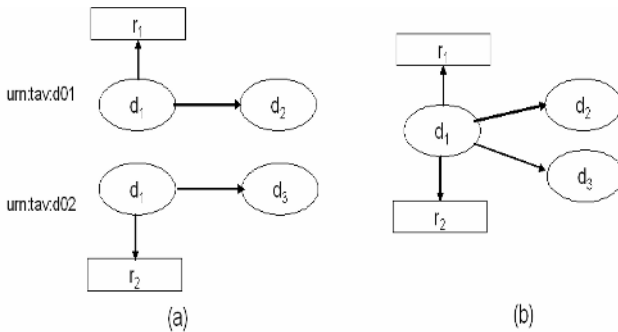


Fig. 4. (a) d_1 identified as `urn:tav:d01` was derived from d_2 in the run r_1 and its equivalent data product d_1 identified as `urn:tav:d02` was derived from a different data product d_3 in the run r_2 . (b) If d_1 is identified by one identity, the data derivation paths for d_1 that were generated in different runs will be converged.

4 Identity Solutions

In order to cope with the *execution* identity problem we propose an identity protocol for building co-references of execution polygonomies. This protocol is

asynchronous to workflow enactment and data and provenance collection. We ignore external polyonomies in this protocol.

4.1 Co-reference Identity Protocol

Definition 2. Consider a data product d , the $IDSet(d) = \{id_1, id_2, \dots, id_n\}$ stores a set of polyonomous identities for d . Each $IDSet(d)$,

- has its own identity: $id(IDSet(d))$;
- is associated with the content of d .

For example we have $IDSet(brpt_1) = \{urn:tav:b0\}$ for $brpt_1$. $IDSet$ objects are stored in the identity store by the identity service in Figure 3. The dashed part in Figure 3 shows the architecture of how the identity protocol functions when executing a workflow in Taverna and interacts with other components in $myGrid$. Three actors participate in the identity protocol: the external service, the workflow enactor and an identity service.

1. The workflow enactor passes data to external services, then invokes these services based on the workflow definition.
2. The service returns the data results and the external identities for gathered data products. When the service returns gathered data products in a response message to the enactor, it should publish identities for these data products in its response message. If the service failed to do this, it would be regarded as a service with lower quality than those that do so.
3. When the data products are returned to the enactor by the service, the enactor assigns Taverna LSIDs to these data products.
4. The enactor invokes the identity service, passing messages to the identity service that include: the data products, their Taverna LSIDs and associated external LSIDs (if any).
5. The identity service intercepts Taverna identities for gathered and computed data products and retrieves or builds $IDSet$ objects for these data. It updates the $IDSet$ objects in the identity store and inserts these $IDSet$ identities as metadata of the data products into the provenance store KAVE. The enactor stores the data products in the data store BACLAVA and their provenance in the provenance store KAVE.

The step of building $IDSet$ objects can be taken offline, by analyzing the provenance store and the data store after the workflow run. A relational identity store is used for keeping the $IDSet$ objects. The RDF KAVE store contains provenance metadata as well as the $IDSet$ metadata for the data products. We name this upgraded KAVE as $KAVE+$. These $IDSet$ metadata are generated by the following scheme and can be used when integrating and comparing provenance graphs, as shown in the Section 4.3.

4.2 Revised Identity Naming Scheme

This identity protocol constructs and updates $IDSet$ objects by three means: (1) identities of gathered data products, (2) data values of computed data products and (3) data objects contained in collection data products at the transport level.

Allocation 1. Allocation by Identities. The identity service receives a *gathered* data product d , either atomic or collection:

1. Search for an existing IDSet object using the external identity of d . If an IDSet is found, retrieve it; otherwise create one.
2. Update the identity store with d 's external and Taverna identities, and its IDSet object identity, $id(IDSet(d))$.
3. Insert this IDSet object and its relationship with d as RDF statements into the provenance metadata store KAVE+.

Allocation 2. Allocation by Values. The identity service receives a *computed* data product d , either an atomic or a collection, e.g. a *brpt*, which collects a set of external data objects:

1. Search for an existing IDSet object using d 's data value. If an IDSet is found, retrieve it; otherwise create one.
2. Update the identity store with the Taverna identity of d and the IDSet object identity, $id(IDSet(d))$.
3. Insert this IDSet object and its relationship with d as RDF statements into the provenance.

Allocation 3. Allocation by Objects. The identity service receives a *transportation collection* data product d , which contains a collection of atomic or collection data products. This d and each of its elements is identified by a Taverna LSID, e.g. the *seqlist₁* containing a collection of atomic gathered data products seq_i in Figure 1.

Definition 3. Consider two collection data products d_i and d_j ($d_i \neq d_j$), d_i and d_j are equivalent, $d_i \equiv d_j$, if and only if both d_i and d_j have the same size, and all elements in them are equal.

For a collection data product d , the identity service should:

1. Search for any existing IDSet objects for each element of d . Search by the element's identity if it is a gathered data product; and search by the element's value if it is a computed data product.
2. Search for d 's equivalent collection data product and an existing IDSet object. If an IDSet is found, retrieve it; otherwise create one.
3. Update the identity store with d 's Taverna identity, and its IDSet object identity, $id(IDSet(d))$.
4. Insert the relationship of d and its element data products into the identity store.
5. Insert this IDSet object and its relationship with the data product d as RDF statements into the provenance metadata store.

This protocol repairs the execution polyonomies in Taverna. Currently, polyonomous identities continue to be allocated in ^{my}Grid to avoid the costs of allocating unique identities for equivalent data products during workflow runs. This protocol and naming scheme, however, improves the maintenance of: (a)

the external identities associated with gathered data products; (b) the relationship between computed data product and its element data products, computed or gathered; and (c) Taverna identity co-references.

4.3 Putting the Identity Service to Use

If equivalent data products are identified by the same identity, the provenance aggregation, integration and comparison will be possible. The IDSet resolves this problem by maintaining a collection of polyonomous identities for a d . The identity of an IDSet object provides a universal identity for a d in ^{my}Grid. An identity service prototype was implemented as a plug-in to Taverna, building IDSet objects for equivalent data products during workflow runs:

- For a **gathered** data product, such as the protein sequence seq_1 , its IDSet object is built by the data product's external identity.
- For a **computed** data product, such as $brpt_1$, the external identities of its elements are parsed and extracted from the data content. $brpt_1$'s IDSet object is built by these external identities of its elements.
- For a **collection** data product at the transportation level, such as the $seqlist_1$, $gbrptlist_1$ in Figure 1, its IDSet object is built by the IDSet identities of its element data products.

To show how the identity service can help us achieve the goal of integrating provenance graphs from repeated runs of every pair-wise runs r_1 and r_2 of WF1, we conduct the following:

1. Aggregate the data provenance graphs. This returns two provenance graphs $P(r_1)$ and $P(r_2)$.
2. Normalize each provenance graph $P(r_i)$. For each data product in $P(r_i)$, retrieve its IDSet identity. If an IDSet identity is found, replace the data identity in $P(r_i)$ with the IDSet identity. This operation returns a normalized graph, $P^n(r_i)$, with each equivalent data product produced in different runs being identified by its IDSet identity.
3. Compare the normalized graph $P^n(r_1)$ and $P^n(r_2)$.

We succeeded in detecting the similarities and differences between $P(r_1)$ and $P(r_2)$. This approach is much faster than identifying equivalent data products at the time of comparison. The number of data objects contained in each provenance graph $P(r_i)$ ($i = 1, 2, \dots, n$) decides the size of $P(r_i)$ and impacts the computational complexity of normalizing a $P(r_i)$. An optimization of this normalization is needed if a large provenance graph is to be processed.

5 Related Work

Polyonomous identities are a common problem in data integration. RefSeq [13] builds cross-references for sequence data across multiple major sequence databases. The Handle system provides GUIDs for digital objects assured by a global

naming authority [14]. This is hard to achieve in life sciences because of the autonomy of data and tool providers. Our identity protocol focuses on constructing co-references between Taverna polyonomies for equivalent data products. These identity co-references can be published at different places and then merged by set calculus. This identity protocol can be adopted in many service-oriented architectures such as the provenance collection architecture proposed by Groth [15].

The Virtual Data Language (VDL) [16] represents derivation relationships between data that were processed by computational procedures. It is unclear whether the aggregation or integration of provenance over repeated reruns of the same workflow over the same data is an issue. The focus is on computed data products and the identity issue becomes one of data mining for identical data values rather than the maintenance of external data object identity and taking care not to allocate polyonomous identities.

In scientific study it is important to harvest provenance logs across runs, initialized by different users or groups [17]. But no naming scheme is defined in up to date provenance work to avoid polyonomous identities for equivalent data products. VisTrails traces the provenance of how workflows are revised from one to another [18]. However, the identity problem for data products remain un-clarified.

If unique identities are allocated for equivalent data products, a mechanism is required to differentiate the different contexts in which each data product are produced. The PASOA project includes a workflow run id as part of the data product id that is produced in a particular workflow run [15]. However, additional contextual information is needed by bioinformaticians, such as the person who produced the data, and an intermediate data product from which a collection of data products were derived from. myGrid adopts named graphs [12] to incorporate more contextual information with data products.

6 Conclusion

In previous work we focused on building a provenance model and the technology to represent this model [2]. The model was developed to assist not only in keeping audit trails of a single workflow run, but also to support the analysis of multiple provenance logs across multiple workflow runs. Although all the provenance graphs are represented as RDF graphs, analyzing these graphs requires a scheme to identify equivalent data products; both common gathered products and corresponding computed products. Allocating and assigning data product identity proved harder than anticipated in practice for provenance graphs that are independently generated, yet need to be combined. Our previous identity allocation scheme works well when we do not aggregate, integrate and compare provenance, but raises issues when we do.

Bioinformatics workflows not only generate new data, but also discover new information by combining and collating existing data. These pre-existing data, external to the Taverna world, have their own identities allocated. In addition, local identities are published for these data products, such as Taverna LSIDs and

VDL's Logical File Names. The autonomous nature of the current bioinformatics domain makes it hard to adopt a global naming service unless a community-wide agreement is achieved, such as the data transfer standard in earth science [19] and in the caBIG project ¹¹. Our problem is particularly acute in that we do not prescribe a closed, strongly typed data environment such as that dictated by caBIG. The multiple identity problems are present in a large extent such as on the Web. Polynomous identities point to the same entity hosted by different web sites. In this paper we tried to enumerate the different identity duplication problems for equivalent data products. To resolve this problem we revise our identity allocation scheme and construct co-references between polynomous identities.

In this paper we aggregate, integrate and compare provenance graphs from repeated runs of the same workflow. We aim also to analyze provenance graphs from runs of *corresponding* workflows. Workflows using different services in the workflow definitions, but realizing the same experiment goal and function correspond. Corresponding workflows evolved one from another. To analyze provenance graphs from runs of corresponding workflows requires a full-fledged definition of the corresponding relationship among workflows and an infrastructure to maintain this relationship, as presented in [18].

Two scalability issues remain to be solved: (1) the scalability of identifying corresponding collection data products computed in iterations in different runs. Iterations over iterations in a workflow run produce collections with multiple hierarchies, i.e. collections containing collections iteratively. This makes it expensive to retrieve corresponding collections with multiple hierarchies; and (2) the scalability of exploiting identities. As shown by the example of exploiting identities in Section 4.3, the computation complexity of normalizing a provenance graph is decided by the size of the provenance graph. This normalization is required in provenance integration and comparison. An optimization is needed if a large provenance graph is to be normalized.

Acknowledgements

The ^{my}Grid project, grant numbers GR/R67743, EP/D044324/1 and EP/C536444/1, is funded under the UK e-Science programme by the EPSRC. The authors would like to acknowledge the other members of the ^{my}Grid team for their contributions, and in particular acknowledge Tom Oinn, Matthew Pocock, Daniele Turi and Chris Wroe. We thank Stian Soiland and David Withers for their useful comments.

References

1. Stevens, R., Tipney, H.J., Wroe, C., Oinn, T., Senger, M., Lord, P., Goble, C., Brass, A., Tassabehji, M.: Exploring Williams-Beuren Syndrome Using ^{my}Grid. *Bioinformatics* **20** (2004) 303–310

¹¹ <https://cabig.nci.nih.gov/>

2. Zhao, J., Wroe, C., Goble, C., Stevens, R., Quan, D., Greenwood, M.: Using semantic web technologies for representing e-science provenance. In: Proc. of the Third International Semantic Web Conference. Volume 3298. (2004) 92–106
3. Li, P., Hayward, K., Jennings, C., Owen, K., Oinn, T., Stevens, R., Pearce, S., Wipat, A.: Association of variations on i kappa b-epsilon with graves' disease using classical and mygrid methodologies. In: Proc. of the UK e-Science AHM. (2004)
4. Oinn, T., Greenwood, M., Addis, M., Ferris, J., Glover, K., Goble, C., Hull, D., Marvin, D., Li, P., Lord, P., Pocock, M.R., Senger, M., Wipat, A., Wroe, C.: Taverna: Lessons in creating a workflow environment for the life sciences. *Journal of Concurrency and Computation: Practice and Experience* (2005) in press.
5. Weisstein., E.W.: (Graph union) <http://mathworld.wolfram.com/GraphUnion.html>.
6. Weisstein., E.W.: (Graph difference) <http://mathworld.wolfram.com/GraphDifference.html>.
7. Clark, T., Martin, S., Liefeld, T.: Globally distributed object identification for biological knowledgebases. *Briefings in Bioinformatics* **5** (2004) 59–70
8. Martin, S., Hohman, M.M., Liefeld, T.: The impact of life science identifier on informatics data. *Drug Discovery Today*. **10** (2005) 1566–72
9. Dalziel, J.: DOI in a DRM environment. White paper, Macquarie University (2004)
10. Altschul, S., Gish, W., Miller, M., Myers, E., Lipman, D.: Basic local alignment search tool. *Journal of Molecular Biology* **215** (1990) 403–410
11. Galperin, M.Y.: The molecular biology database collection: 2006 update. *Nucl. Acids Res.* **34** (2006) 3–5
12. Carroll, J., Bizer, C., Hayes, P., Stickler, P.: Named graphs. *Journal of Web Semantics* **3** (2005)
13. Pruitt, K.D., Maglott, D.R.: Refseq and locuslink: Ncbi gene-centered resources. *Nucleic Acids Research* **29** (2001) 137–140
14. Kahn, R., Wilensky, R.: A framework for distributed digital object services. Technical Report tn95-01, Macquarie University (1995)
15. Groth, P.T., Luck, M., Moreau, L.: A protocol for recording provenance in service-oriented grids. In: Proc. of the Eighth International Conference on Principles of Distributed Systems, Grenoble, France (2004) 124–139
16. Foster, I., Vockler, J., Wilde, M., Zhao, Y.: The virtual data grid: A new model and architecture for data-intensive collaboration. In: Proc. of the First Biennial Conference on Innovative Data System Research. (2003)
17. Futrelle, J.: Harvesting rdf triples. In: Proc. of the Third International Provenance and Annotation Workshops. (2006) in press.
18. Bavoil, L., Callahan, S.P., Crossno, P.J., Freire, J., Scheidegger, C.E., Silva, C.T., Vo, H.T.: Vistrails: Enabling interactive multiple-view visualizations. In: Proc. of IEEE Visualization. (2005) 135–142
19. Arctur, D.K., Hair, D., Timson, G., Martin, E.P., Fegeas, R.: Issues and prospects for the next generation of the spatial data transfer standard (SDTS). *International Journal of Geographical Information Science* **12** (1998) 403 – 425

CombeChem: A Case Study in Provenance and Annotation Using the Semantic Web

Jeremy Frey¹, David De Roue², Kieron Taylor¹, Jonathan Essex¹,
Hugo Mills², and Ed Zaluska²

¹ School of Chemistry, University of Southampton, Southampton, SO17 1BJ, UK
{j.g.frey, krt1, j.w.essex}@soton.ac.uk

² School of Electronics and Computer Science,
University of Southampton, Southampton, SO17 1BJ, UK
{dder, hrm, ejz}@ecs.soton.ac.uk

Abstract. The CombeChem e-Science project has demonstrated the advantages of using Semantic Web technology, in particular RDF and triplestores, to describe and link diverse and complex chemical information, covering the whole process of the generation of chemical knowledge from inception in the synthetic chemistry laboratory, through analysis of the materials made which generates physical measurements, computations based on this data to develop interpretations, and the subsequent dissemination of the knowledge gained. The project successfully adopted a strategy of capturing semantic annotations ‘at source’ and establishing schema and ontologies based closely on current operational practice in order to facilitate implementation and adoption. The resulting ‘Semantic Data Grid’ comprises around 45 million RDF triples across multiple stores.

1 Introduction

This paper reports on our experiences in building a Semantic Web infrastructure for chemical research as part of the CombeChem project funded by the U.K. e-Science programme. We set out to use the available Grid and semantic technology to support the entire chemical research sequence. This typically starts from an experiment producing data, which is then searched for relevant patterns, which lead to results, conclusions and publications and which in turn leads to further experiments. All progress depends on individual scientists building on the results already produced by others.

While in the past this process has served the science community well, it is now in danger of paralysis from the sheer quantity of data being produced [1]. We considered it essential that semantic support be introduced at every stage in the process to facilitate automated mechanisms for research support, providing an infrastructure where complex applications and services can be deployed with minimal manual intervention [2,3]. This level of automation is required to deal with the increasing rate at which scientific data is being generated and needs to be processed if the integration of the data and its transformation into information and knowledge is to keep pace with the generation of the data. We describe our solution as a “Semantic Data Grid” as considered from a Grid perspective in [4].

Our architecture has been designed to enable effective capture of both data and metadata at the earliest opportunity during the scientific investigation. [5] Once captured the material is maintained and organized as it traverses the virtual organisation that represents the whole Chemistry community involved in converting a new piece of data into accepted chemical facts and knowledge. Our methodology has been to draw as far as possible on established chemistry practices and then augment them. Our principle is that by constructing schema and ontologies based on current operational practice we have a solution that is known to work, and we regard this as an essential first step to facilitate deployment and adoption.

In section 2 we explain our use of Semantic Web, leading us to the Semantic Data Grid. We then go on to explain our implementation and experiences in Section 3, illustrate the system by focusing on two parts – the “Smart Lab” and the large store of chemical descriptions. The conclusions and future work form Section 4.

2 Semantic Web Approach

To enter this semantically-described world we adopted a policy that we call “annotation at source”, recognising that the digital world necessary to implement our vision must start as soon as possible in the information chain. For example, the CombeChem project set out to support the relatively small-scale needs of everyday scientists in recording experiments. This resulted in an interest in the Electronic Laboratory Notebook (ELN) research area as part of the overall concept to provide effective semantic support for experimental and computational science. Here the “annotation” starts in the safety plan before the experiment has even begun.

We set out to provide comprehensive semantic support for the whole spectrum of chemistry research in as transparent a fashion as possible, adopting RDF to capture the semantic content and describe the scientific data. A fundamental part of this strategy was to study established practice in the field and to introduce as few changes to normal everyday working practice as possible. This was part of the objective of capturing as much metadata at source (i.e. as it is generated), completely automatically. We adopted the additional premise that it would be impossible to predict in advance the way that data would be accessed and used, hence flexibility of use was a fundamental objective. This led directly to the requirement that the information infrastructure to hold this data and metadata should be as general as possible. We adopted the InChI (a character string that uniquely describes an organic molecule based on its structure) as our shared identifier [6].

In summary our design approach adopted four principles:

1. Grounding in established operational practice – our starting point is to study chemists at work;
2. Capturing a rich set of associations between all types of things, expressed pervasively in RDF and hence explicitly addressing the sharing of identifiers;
3. Metadata capture should be automated as far as possible – our goal is augmentation not disruption;
4. Information will be reused in both anticipated and unanticipated ways.

Other approaches to the problem include the World Wide Molecular Matrix [7], while the Collaboratory for Multi-Scale Chemical Science [8] is developing an informatics-based approach to synthesising multi-scale information to create knowledge.

3 Implementation

3.1 Smart Lab

The back-end of the Smart Lab [9] system consists of a database which stores the details of the experiments, and presents a query interface for interrogating the datatore. The data for the experiment is stored in an RDF triple store based on Jena [10], and is accessed by the front-end applications through a SOAP-based query interface.

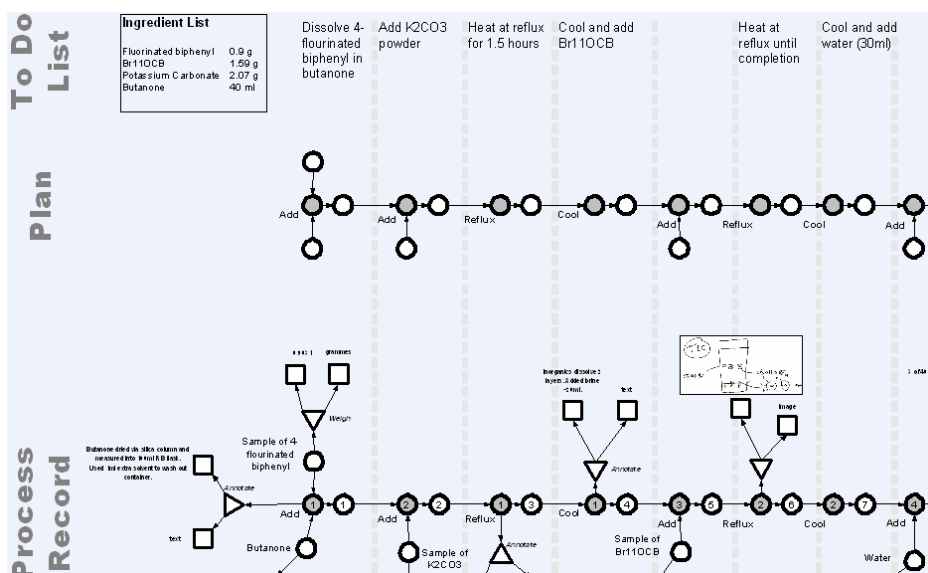


Fig. 1. Fragments of the RDF graph, showing the distinction between Plan and Record

We have developed three primary applications: a planning tool, which is used to set up the plan and ingredients for the experiment; a weigh-station/liquid-measure application, used for recording the quantities of ingredients actually used, as an example of a measurement device; and a "bench" application, used for making notes and annotations on the plan while performing the experiment. The methodological approach is described in [11]. The latter two applications we have implemented on a Tablet PC, to be carried around in the laboratory. The current prototype planner application is implemented as a set of dynamic, form-based web pages. The "smart lab" system is modular. For instance, other measurement devices, such as a digital camera or a formatter for adding mass spectrograph recordings, can also be added to the system in the same way as the weigh-station application.

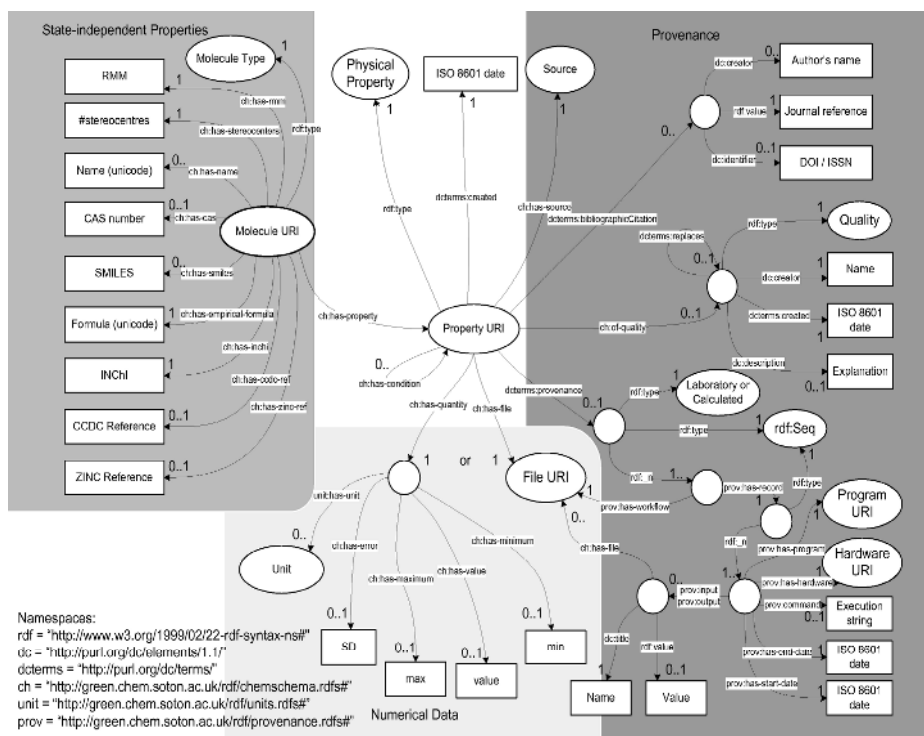


Fig. 2. The schema for the CombeChem Semantic Data Grid

The functional core of the SmartLab software is the libtea library. This library provides abstraction of the low-level RDF data structures kept in the triple store, and queried and served by the ModelServer. The programming interface (API) for libtea abstracts away some of the details of the underlying RDF structures from the programmer, and presents a simpler interface for the "standard" structures and properties encoded in the base SmartLab ontology. It does not, however, explicitly hide the RDF from the programmer, so if it is necessary to add new features or structures to the schema, it is possible to do so in a flexible manner without having to add support directly to libtea. The libtea API presents a set of objects to the programmer, each object representing a different concept within the RDF structure, and encapsulating a subgraph of one or more triples.

3.2 Triple Store

The digital record from the Smart Lab data then feeds into the scientific data processing. The creation of original data is accompanied by information about the experimental conditions in which it is created. There then follows a chain of processing such as aggregation of experimental data, selection of a particular data subset, statistical analysis, or modelling and simulation. The handling of this information may include explicit annotation of a diagram or editing of a digital image. All manipulation of the

data through the chain of processing is effectively an annotation upon it and the provenance is explicit. The annotations are required to be machine processable, and useful for both their anticipated purpose and interoperable to facilitate subsequent unanticipated reuse. This is achieved by RDF being used through the system. At the time of writing there are 45 million RDF triples in the Combechem triplestore. The current target is about ten times this number, representing a very substantial Semantic Web deployment.

Figure 2 shows the schema for the CombeChem data grid, based around chemical properties. Objects are marked as ellipses, the arrows show how predicates link objects together, and rectangles are literal values.

We evaluated several triplestores and adopted 3store [12] because it has good scaling properties. Additionally it is easily batch or perl scriptable, supports RDFS, and it can use RDBMS tools for maintenance of data (e.g. backups and migration) as all application state is held in the database. 3store supports the SPARQL query language.

We have succeeded in feeding approximately 80 million triples into the triplestore. Queries remain responsive, but data import performance has begun to degrade, i.e. stores is known to be a challenging issue and we can envisage how a single large triplestore with frequent insertions would be unable to cope with potential demand. 80 million triples equates to a reasonably-sized chemical dataset, but could easily be

2-Benzyl-4',6'-bis(trifluoromethyl)-6'-hydroxy-2',3',7',7a'-tetrahydro-6'H-spiro-(furan-3,7'-(furo(3,2-c)pyran))

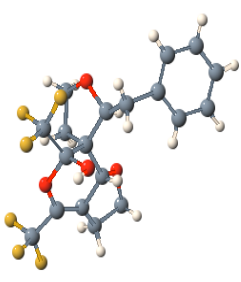
S. J. Coles, J. M. Mellor, A. H. El-Sagheer, E. E. -D.
M. Salem and R. N. Metwally.

University of Southampton

$C_{19}H_{18}F_6O_4$

CCDC Code: WOSSUS
ICHI Code: C19H18F6O4,20-17(21,22)13-12-6H2-8H2-28-15H(12)16(18(26H,29-13)19(23,24)25)7H2-9H2-27-14H(16)10H2-11-4H-2H-1H-3H-5H-11
[\(google for ichi\)](#)

Compound Class: Organic
Keywords: additions; Grignard reagents; trifluoromethylketones
Creation Date: 13 September 1999
Deposited By: [Susanne L. Huth](#)
Deposited On: 30 July 2004



Available Files

Final Result

Data collection parameters	Chemical formula	C19 H18 F6 O4	99sot029_data/99sot029.CIF	14k
			99sot029_data/99sot029.cml	7k

Fig. 3. The eCrystals interface

doubled or trebled when populating with computed properties. Hence we are now contemplating alternative ways of partitioning and maintaining the triples across multiple stores. Progress is also now being made in linking the RDF structures for the molecular properties and those describing the experiments from the ELN, tied together by the molecules URI.

Figure 3 illustrates one interface to this data, developed in the eBank project [13]. The information contained within an entry in this archive is all the underlying data generated during the course of a structure determination from a single crystal x-ray diffraction experiment. An individual entry consists of three parts: core bibliographic data, such as authors, affiliation and a number of chemical identifiers; data collection parameters that allow the reader to assess at a glance certain aspects of the crystallographic dataset; files available for download (visualisations of the raw data, the raw data itself, experimental conditions, outputs from stages of the structure determination, the final structural result and the validation report of the derived structure).

4 Conclusions and Future Work

RDF has been shown to be an effective method for capturing highly detailed chemical data, allowing it to be indexed in a persistent triplestore such that it can be searched and data-mined in useful ways. The triplestore has now reached a viable state with further addition of chemical properties as an ongoing process. We are now beginning to develop automated calculations using the many available structures and to store the results alongside all the details of the computations that produced them. Beyond that we can achieve high-throughput data processing and begin to develop new models based on those computations.

The ELN proves to be an excellent model system for the meeting of the semantic chemical grid descriptions of materials and services with the pervasive environment needed to capture the information within the source laboratory. We are currently conducting more investigations in the use of the smart lab systems in an active synthetic organic chemistry laboratory looking at the use and re-use of information captured using our systems. The studies will soon be extended to investigate the pervasive aspects of the grid looking at the use of handheld systems (e.g. PDA, tablets) systems vs. distributed computers in different positions within the laboratory.

More work is required to build up the chemical ontology to the level comparable with the XML structure provided by CML. In the INCHI we have a computable URI for organic molecules, but this leaves large areas of compounds (inorganic, mixtures, materials etc.) without adequate URIs of this form. We have seen in the need to provide an RDF structure for units that the process of describing a piece of scientific data, with all the necessary descriptions and provenance, propagates requirements out in an extensive net, meeting though with other domains, where we can link up with other semantic descriptions. An area which demands rapid attention because of the importance of unhindered and accurate information flow between different knowledge domains, is the need to integrate the chemical ontology with for example the LSI identifiers is a part of the whole processes of linking Bio- and Chemical Informatics, to aid for example drug modelling (sample and model selection in QSAR) and on to

the larger spatial scale of the environment, all of which are major purposes of our current investigations.

We have also commenced an exploration of capturing scientific discourse within the Data Grid, fully linked in following the publication@source approach. This includes materials from meetings and videoconferences, and is achieved through the use of meeting support tools which capture semantic annotation following a similar approach to the smart lab. [14]

The importance of provenance cannot be overstated – not just with respect to understanding where information has come from, but in understanding how to interpret it. An item of information is rendered almost useless if the details of the provenance are not known (in practice this leads to experiments being unnecessarily repeated).

Our principle of pragmatism has brought us a long way – this is a significant piece of the Semantic Web. In a sense, we are now ready to begin! We have benefited from flexible associations and from the sharing of identifiers, and from graph queries and chaining in the triplestore. Much of the power of the Semantic Web that comes from ontologies is yet to be explored. Also at this level we see opportunities for use of rules as these solutions emerge within the Semantic Web stack.

Acknowledgements

The work in this paper was partially supported by the UK e-Science CombeChem project (GR/R67729/01), the Advanced Knowledge Technologies IRC (GR/N15764/01), CoAKTinG (GR/R85143/01) and and Semantic Media (EP/C010078/1). eBank is funded by JISC.

References

1. Hey and Trefethen, Cyberinfrastructure for e-Science, *Science* 308 (2005) 817-821.
2. C. A. Goble, D. De Roure, N. R. Shadbolt, and A. A. A. Fernandes, "Enhancing Services and Applications with Knowledge and Semantics," in *The Grid 2: Blueprint for a New Computing Infrastructure*, I. Foster and C. Kesselman, Eds.: Morgan-Kaufmann, pp. 431-458. (2004).
3. De Roure, D. Jennings, N.R. Shadbolt, N.R. The Semantic Grid: Past, Present, and Future, *Proceedings of the IEEE*, 93, (2005) 669-681.
4. Taylor, K., Gledhill, R., Essex, J.W., Frey, J.G., Harris, S.W. and De Roure, D. "A Semantic Datagrid for Combinatorial Chemistry", *Proceedings of IEEE Grid Computing Workshop at SC05*, IEEE, Seattle, WA. November 2005.
5. Taylor, K., Essex, J.W., Frey, J. G., Mills, H. R., Hughes, G and Zaluska, E. J. The semantic grid and chemistry: experiences with CombeChem. *Journal of Web Semantics* (In Press).
6. Taylor, K. R., Gledhill, R., Essex, J. W., Harris, S.W., De Roure, D. C. and Frey, J. G. Bringing chemical data onto the semantic web. *Journal of Chemical Information and Modeling*, 46 (2006),939-952. (doi:10.1021/ci050378m)
7. Frey, J.G., Bradley, M. Essex, J., Hursthouse, M.B., Lewis, S.M., Luck, M., Moreau, L., De Roure, D., Surridge M., and Welsh, A., 'Combinatorial Chemistry and the Grid', published in 'Grid Computing: Making the Global Infrastructure a Reality', edited by F.Berman, G.Fox and T.Hey, Wiley, 2004.

8. InChI International Chemical Identifier <http://www.iupac.org/inchi/>
9. Murray-Rust, P. "The World Wide Molecular Matrix - a peer-to-peer XML repository for molecules and properties," presented at EuroWeb2002, Oxford, UK, 2002.
10. Collaboratory for Multi-Scale Chemical Science (CMCS) <http://cmcs.org/>
11. Frey, J. G. (2004) Dark lab or smart lab: the challenges for 21st century laboratory software. *Organic Process Research & Development*, 8, (2005) 1024-1035. (doi:10.1021/op049895g)
12. Hughes, G., Mills, H., De Roure, D., Frey, J.G., Moreau, L., schraefel, m.c., Smith G., and Zaluska, E., 'The semantic smart laboratory: A system for supporting the chemical e-Scientist', *Org. Biomol. Chem.* 2, (2004) 3284-3293, 2004.
13. Jena – A Semantic Web Framework for Java, <http://jena.sourceforge.net/>
14. schraefel, m.c., Hughes, G., Mills, H., Smith, G., Payne T., and Frey, J., 'Breaking the Book: Translating the Chemistry Lab Book to a Pervasive Computing Environment', published in *Proceedings of the Conference on Human Factors (CHI)*, 2004.
15. Harris, S and Gibbins, N.3store: Efficient Bulk RDF Storage. In *Proceedings of the First International Workshop on Practical and Scalable Semantic Web Systems (PSSS2003)*, Sanibel Island, Florida, USA.
16. Duke, M., Day, M., Heery, R., Carr L., and Coles, S.J. "Enhancing access to research data: the challenge of crystallography". *Proceedings of the 5th ACM/IEEE-CS joint conference on Digital libraries*, Denver, CO, USA. (2005) 46 – 55.
17. Coles, S.J., Frey, J.G., Hursthouse, M.B., Light, M.E., Milsted, A.J., Carr, L.A., De Roure, D., Gutteridge, C.J., Mills, H.R., Meacham, K.E., Surridge, M., Lyon, E., Heery, R., Duke, M. and Day, M. An E-Science Environment for Service Crystallographys from Submission to Dissemination. *Journal of Chemical Information and Modeling* 46 (2006) 1006-1016, (doi:10.1021/ci050362w)
18. Coles, S., Frey, J. G., Hursthouse, M. B., Light, M. E., Meacham, K. E., Marvin, D. J. and Surridge, M. ECSES - examining crystal structures using `e-science': a demonstrator employing web and grid services to enhance user participation in crystallographic experiments. *J. Appl Cryst*, 38 (2005) 819-826. (doi: 10.1107/S0021889805025197)
19. Bachler, M. S., Buckingham Shum, S. J., De Roure, D. C., Michaelides, D. T. and Page, K. R. Ontological Mediation of Meeting Structure: Argumentation, Annotation, and Navigation. In *Proceedings of 1st International Workshop on Hypermedia and the Semantic Web (HTSW2003)*, Nottingham, 2003.

Principles of High Quality Documentation for Provenance: A Philosophical Discussion

Paul Groth, Simon Miles, and Steve Munroe

School of Electronics and Computer Science
University of Southampton
Highfield, Southampton SO17 1BJ, United Kingdom
{pg03r, sm, sjm}@ecs.soton.ac.uk

Abstract. Computer technology enables the creation of detailed documentation about the processes that create or affect entities (data, objects, etc.). Such documentation of the past can be used to answer various kinds of questions regarding the processes that led to the creation or modification of a particular entity. The answer to such questions are known as an entity's provenance. In this paper, we derive a number of principles for documenting the past, grounded in work from philosophy and history, which allow for provenance questions to be answered within a computational context. These principles lead us to argue that an interaction-based model is particularly suited for representing high quality documentation of the past.

History is important: in order to make progress in the future, it is important to learn the lessons of the past. History, therefore, becomes a vital resource for progressive societies. History is based on evidence or documentation of events that occurred in the past. Computer technology enables documentation to be produced that is more accurate and comprehensive than previously possible. Given the quantity of documentation that can be generated by computer systems (for example those running e-Science applications), what kind of documentation should be created that enables historians and users to most effectively answer questions they have about the past. To answer these questions, we argue for two principles for the creation of, what we term, *high quality documentation* of the past. One principle guarantees that users of documentation of the past have a precise semantics for it. The other principle guarantees that there exists a link between documentation and its creators. Together these principles are the first contribution of this paper.

A question that is often posed by historians and users regards the *provenance* of an entity (object, data item, etc.): what was the process that led to the entity in question? To enable the answering of provenance questions in a manner that conforms with the aforementioned principles, we introduce a model of documentation based on the exchange of messages between actors. The justification of this model is the second contribution of this paper.

The rest of this paper is organised as follows. We begin with a more detailed motivation of our work. After which, three assumptions about the world are

presented. We then present two principles of high quality documentation. Each principle is grounded in work from philosophy and history. Finally, we argue for the adoption of a model, based on interactions between computational programs, as a representation of past documentation that fits both provenance and the principles outlined.

1 Motivation

Historical records are typically produced by historians weaving together scattered pieces of documentation to tell a story of past events and their relationships with one another [10]. A history of ancient Greece, for example, would be pieced together from archaeological sites, preserved texts and artefacts. These pieces of documentation are distributed across the globe at various locations including museums, archaeological digs and libraries. Furthermore, the documentation is far from complete, and the many gaps and omissions must be filled with suppositions and educated guesses. Today, human activity is becoming increasingly automated with the aid of computational systems, which perform part or, in some cases, all of processes previously undertaken solely by humans. Interestingly, this means that activities in scientific [2], governmental [4] and commercial [12] settings are increasingly represented in computational systems, which presents an opportunity: it becomes possible to capture what happens in these settings both accurately and comprehensively.

The large amount of documentation that can be generated by computer systems changes the role of the historian, from someone who deduces history from paltry evidence to one who sifts through detailed documentation in order to make sense of history for others. If historians are to play this role, how can we ensure their task is simplified and facilitated? More specifically, how can computer scientists provide the correct kinds of documentation to facilitate the description and analysis of processes that have occurred and are captured within computational systems?

In this paper, we answer these questions for documentation about the functioning of computer systems. Furthermore, we take into account a specific type of historical question: *provenance*. Answering provenance questions about entities places requirements on the documentation of the past. In order to refer to the history of an entity as it existed at a point in time, or in the context of a specific event, documentation of the processes the entity goes through must exist up to and including that point in time. For instance, to track the provenance of a Renaissance painting, the owner needs to know both its owner in 1990 as well as its owner in 1800 and, preferably, all of the owners in between. The documentation must also be able to include information from several sources. Referring to our painting example, one source may give the ownership information for a painting in 1800, while another may provide it for 1990. Considering these requirements within computational systems, we derive a number of principles of documentation of the past that make it possible to find the provenance of data items. We term documentation of the past that conforms to these principles

high quality documentation, which is the kind of documentation that we believe historians and users should be sifting through as they answer questions about the provenance of data items.

We now argue for our principles of high quality documentation of the past.

2 Three Basic Assumptions

The set of principles we intend to outline are based on a number of assumptions about the world. We believe these assumptions to be reasonable, but given that they are philosophical in nature, they are open to debate. We abstain from such debate here and instead encourage the reader to consult the philosophical literature (such as [11]).

We postulate that historians as well as others take a realist view of the world in everyday life. A general definition of realism is given in our first assumption:

Assumption 1 (Realism). *“a, b, and c and so on exist, and the fact that they exist and have properties such as F-ness, G-ness, and H-ness is independent of anyone’s beliefs, linguistic practices, conceptual schemes, and so on.” [5]*

That is to say, things like computers, paper and post-it notes as well as their attributes of being rectangular, made of wood, yellow in colour are independent of what anyone thinks of them. People tend to believe that cars, post-it notes, and magazines exist without having to be there to see them. Extending this concept, we introduce the assumption of verification.

Assumption 2 (Verification). *Observers can check the existence of all entities and their properties independently from other observers.*

The implication of this assumption is, for example, if an observer claims that there is a large red dinosaur outside their office window, another observer who is there at the same time can check whether this is the case or not. The ability to verify is dependent on two things: the independence of the environment from the observer (a consequence of realism) and the observers’ senses not malfunctioning.

Our final assumption is as follows:

Assumption 3 (Truthful representation). *Users of documentation of the past, such as historians, want documentation to be an accurate or truthful representation of what occurred.*

In essence, the more detailed, accurate, and truthful portrait of the past given, the more users’ analyses can be supported and buttressed. With these assumptions in mind, we now describe principles for high quality documentation. We begin with a discussion of truth.

3 Truth and Factuality

As stated in Assumption 3, we assume that users want documentation of the past to be truthful. To establish that documentation does indeed truthfully represent

the past, we must find a definition of truth that works with respect to both computational systems and the past. We begin by presenting a theory of truth that fits our assumptions. We then show how the assumption of verification is broken with respect to computer systems. The verification assumption is then discarded and we develop a new principle, based on the theory of truth we present, that provides documentation of the past for computational systems with a precise semantics.

From Assumption 1, the correspondence theory of truth seems to be the standard defensible theory of truth that we should adopt [9]. It is defined as follows:

Definition 1 (Correspondence theory of truth). *“A proposition is true just in case (if and only if) it corresponds to fact or the world.” [9]*

That is to say, if a statement is expressed and it corresponds (or can be mapped) to the world, then it is true.¹ The correspondence theory means that statements can be verified by observing what they correspond to. Assumption 2 holds in the real world. Humans can check if statements are true by observing the world. They can observe whether or not a large red dinosaur is outside the office window.

In computer systems, however, the assumption of verification does not work. Software programs or components, which we call actors, do not have independent access to a common element (piece of hardware, software, component). There is no equivalent to the independently accessible environment present in the physical world. Instead, actors rely on information communicated between themselves. For example, to access the hard drive, an application program communicates with the operating system, which communicates with the driver program which accesses the physical drive and provides the data back through the chain. The application program’s access to the hard drive is mediated by other programs. It does not have independent access to the hard drive. The only actor that can know directly the contents of the hard drive is the driver program. If it receives six different requests and provides the same data back for every request, no other actor would be able to detect if that was an incorrect or correct response.

In distributed computational systems, actors’ dependence on communication is further emphasised by their spatial isolation. For an actor executing in computer A to know about the state of computer B, it must receive information from an actor executing in computer B. There is no other way for an actor in A to gain direct knowledge of computer B. Therefore, unlike a human, an actor cannot readily verify the state of the world independently of the information it receives from other actors. If an actor executing on computer A makes a proposition about the state of computer A, even if that proposition corresponds to the world, an actor on another computer cannot verify the truthfulness of the proposition. Hence, the assumption of verification does not hold in a computer system.

Given this problem, we make explicit what actors can verify to be truthful. Actors come to know the world via communication. They observe the world through the receipt of information. Therefore, an actor can determine whether

¹ Throughout this paper, we will use “statement” and “proposition” interchangeably.

a proposition is true with respect to what it observes. The statement “actor A received data item X” can be verified by actor A as being true or false. However, no other actor can verify that statement. In a computational system, the verification of a statement is dependent both on its correspondence to a fact and the actor that observed the fact.

Therefore, the correspondence theory of truth still holds in computational systems except that only an actor that has observed a particular event can know whether a proposition about it is true. This implies that other actors as well as users cannot know whether or not documentation about what happened in computer systems is true. Hence, we are led to the notion that documentation cannot be independently verified as truth. Instead, for every statement in documentation of the past, a user must make a judgement about its veracity. To enable this sort of judgement we introduce the following principle.

Principle 1 (Factuality). *As part of documentation of the past, actors must only record propositions that they can verify to be true, where truth is defined by the correspondence theory of truth.*

Actors, then, should only make statements about what they observe, but not about guesses or inferences about the world. If this principle is followed, users of documentation can know that statements represent reality at that time for the actors who made them. This is a powerful notion. Documentation created using Principle 1 can be interpreted as *representing the reality of the computer system* assuming that the users believe the actors that created the documentation. This notion is currently not enforced by most provenance systems. Scientific Annotation Middleware [7], for example, allows actors to record inferences about what other actors have done.

4 He Said, She Said

In the previous section, we stated that users should interpret documentation as statements by actors in computer systems about what they have observed. Documentation about the past then acts as *evidence* that the past occurred in some manner. Evidence plays a critical role in society. Historians, juries and others rely on evidence to make judgements about the past and predict what will happen in the future. In a legal setting, evidence is defined as follows:

Definition 2 (Evidence). *“Evidence is information, whether in the form of personal testimony, the language of documents, or the production of material objects that is given in a legal investigation, to establish the fact or point in question” (Oxford English Dictionary)*

We now draw a parallel between the statements that make up the documentation of the past for a computer system and a particular type of evidence in a legal setting, *testimony*. Coady (p. 33) gives a six point list of how testimony in a legal setting can be identified [1]. We enumerate the most pertinent here.

1. Testimony is a form of evidence.
2. Testimony is constituted by persons A offering their remarks as evidence so that we are invited to accept p because A says that p .
3. The person offering the remarks is in a position to do so, i.e. he has the relevant authority, competence, or credentials. Within English law and proceedings influenced by it, the testimony is normally required to be firsthand (i.e. not hearsay).

The statements made by actors are similar to testimony. They are evidence that something happened in a computer system. We are invited to accept a statement by an actor because the actor states it. Furthermore, the actor, from the principle of factuality, should have firsthand knowledge of what occurred. Just like eyewitnesses to a crime scene testifying in court, actors provide statements as to how things were at a particular time, hopefully without inference. However, just like testimony from people, statements made by actors may be incomplete, inaccurate or misconstrued.

Users of documentation of the past must then play a similar role to juries. Just as juries make a judgement about whether to believe the set of claims provided by an eyewitness, users must make the same judgement about documentation of the past. Such judgements are usually based on the source of the evidence. Users can interpret documentation based on a variety of factors: e.g. what other actors state about the source, the actor's past performance, the content of the documentation.² The key to making this judgement is to know the creator of the statement. Therefore, it is fundamental that documentation about the past in computer systems be attributable, which is our second principle.

Principle 2 (Attribution). *Each statement making up documentation of the past for a computer system must be attributable to a particular actor.*

We have argued that documentation of the past for computer systems should be both attributable and factual. We now endorse a particular model for representing documentation of the past.

5 Endorsing a Model

We have developed mechanisms to record documentation about the processes executing in computational systems [3]. These mechanisms adopt a specific model centered on message exchanges, termed *interactions*. The choice of an interaction based approach was influenced by Milner [6]. We now argue that an interaction model that follows the aforementioned principles is best suited for representing documentation. The interaction model is defined as follows.

Definition 3 (Interaction Model). *In the interaction model, a system is composed of actors that exchange information via interactions. An interaction*

² A user is determining whether to trust an actor. The concept of trust and related research are outside the scope of this paper. We refer the reader to [8] for more in-depth discussions of trust in computational systems.

consists of one actor sending a message and another receiving the same message. Actors receive no data other than via interactions, i.e. there is no external environment.

Following Milner [6], we argue that any computational system can be described using the concepts of the interaction model. From this model, we can define a form of documentation that describes a system according to that model.

Definition 4 (Interaction Model Documentation). *Documentation of the past that describes a system according to the interaction model is called interaction model documentation.*

From the principle of factuality, actors should only document what they observe. This means that the documentation produced by one actor will be created independently from that produced by any others. Given that determining the provenance of a data item, which can include events spanning time and space, may require examining documentation from multiple actors, we need to know when multiple actors are providing documentation of the same event. This is also true more generally, wherever a historian attempts to create an independent narrative from multiple actors' accounts [10]. According to Section 2, the observations made by actors cannot, in themselves, be independently verified but this does not prevent a historian from *inferring* that multiple actors' observations are of the same or connected events from the content of the documentation.

There are two ways that a connection between actors' observations can be inferred from interaction model documentation. First, in some cases, something about where the content of an actor's observations came from can be inferred from that content. For example, the port which an actor received TCP/IP communication on may be apparent from the documentation of that communication. Moreover, in some cases the actual actor that the content came from can be inferred, e.g. a message may be digitally signed so the author can be determined. In this way, we can infer a connection between the data received by one actor and the observations of another actor (the sender).

Second, we can infer explicit connections between actors' communications. In a system of actors with no shared world to observe, connections between actors' observations can only be made if it is apparent when some data was *sent* by an actor over the communication medium. Since observation is the receipt of data, an actor will observe, and so can document, *receiving* a message from another actor but will not directly observe the sending of a message. Fortunately, some observations can be inferred as indirectly documenting information leaving an actor. For example, the content of documentation can imply that data is about to be sent by the actor over the wire, with the intention of reaching another actor. Where an actor receives *feedback* that it is sending or has sent data, it can document this feedback and a historian can infer that it sent the data. Given this, it can be inferred that where actor *A* recorded receiving data, actor *B* recorded (feedback on) sending data, the data has identical content and each message communicated can be safely assumed to be unique, then we can infer that the actors' observations are of the same message.

The interaction model connects what would be, in an event-based model, for example, isolated, disconnected and unrelated events into connected, related and meaningful observations that, taken together, allow descriptions of coherent processes.

6 Conclusion

In this paper, we have proposed two principles that documentation about the past for computational systems should abide by to be considered *high quality*. First, we argued for creating factual documentation so that users have a precise semantics in which to interpret documentation. The argument was grounded in a philosophical investigation into what would make for a truthful representation of the past. Second, based on the observation that statements made by computational actors are equivalent to testimony in a court of law, we derived the necessity for attributable documentation. Finally, an interaction model was endorsed as an effective computational model for representing the past, especially in comparison with event based models.

If historians or any users are to effectively use documentation of the past for provenance, they must understand the underlying principles that were used in the generation of it. Therefore, it is critical that the community enumerate and justify these principles. This paper is the first to state such principles explicitly.

Acknowledgements. This research is funded in part by EPSRC PASOA project (GR/S67623/01) and the European Community's Sixth Framework Program's EU Provenance project (IST511085).

References

1. C. Coady. *Testimony: A Philosophical Study*. Oxford University Press, 1992.
2. J. Freire, C. T. Silva, S. P. Callahan, E. Santos, C. E. Scheidegger, and H. T. Vo. Managing rapidly-evolving scientific workflows. In L. Moreau and I. Foster, editors, *International Provenance and Annotation Workshop (IPAW'06)*, May 2006.
3. P. Groth, M. Luck, and L. Moreau. A protocol for recording provenance in service-oriented grids. In *Proceedings of the 8th International Conference on Principles of Distributed Systems (OPODIS'04)*, Grenoble, France, Dec. 2004.
4. S. Ivarez, J. Vazquez-Salceda, T. Kifor, L. Z. Varga, and S. Willmott. Applying provenance in distributed organ transplant management. In L. Moreau and I. Foster, editors, *International Provenance and Annotation Workshop (IPAW'06)*, May 2006.
5. A. Miller. Realism. In E. N. Zalta, editor, *The Stanford Encyclopedia of Philosophy*. Fall 2005.
6. R. Milner. Elements of interaction: Turing award lecture. *Communications of the ACM*, 36(1):78–89, 1993.
7. J. Myers, A. Chappell, M. Elder, A. Geist, and J. Schwidder. Re-integrating the research record. *IEEE Computing in Science & Engineering*, pages 44–50, 2003.
8. D. H. S. D. Ramchurn and N. R. Jennings. Trust in multiagent systems. *The Knowledge Engineering Review*, 19(1):1–25, 2004.

9. F. F. Schmitt. *Truth: A Primer*. Westview Press, 1995.
10. M. Stanford. *An Introduction to the Philosophy of History*. Blackwell Publishers, 1998.
11. E. N. Zalta. The Stanford Encyclopedia of Philosophy. <http://plato.stanford.edu/>, Spring 2006. ISSN 1095-5054.
12. J. Zhao, C. Goble, and R. Stevens. An identity crisis in the life sciences. In L. Moreau and I. Foster, editors, *International Provenance and Annotation Workshop (IPAW'06)*, May 2006.

Author Index

- Altintas, Ilkay 118
Álvarez, Sergio 28
- Barga, Roger S. 1
Barney, Oscar 118
Bose, Rajendra 193
Bourilkov, Dimitri 19
Bowers, Shawn 133
Branco, Miguel 55
Braun, Uri 171
Buneman, Peter 162
- Callahan, Steven P. 10
Chapman, Adriane 162
Cheney, James 162
Cohen, Shirley 133
- Davidson, Susan B. 133
De Roue, David 270
Deelman, Ewa 90
Digiampietri, Luciano A. 1
Dvořák, František 246
- Essex, Jonathan 270
- Foster, Ian 148
Freire, Juliana 10
Frey, Jeremy 270
Futrelle, Joe 64
- Gannon, Dennis 222
Garfinkel, Simson 171
Gil, Yolanda 90
Goble, Carole 254
Golbeck, Jennifer 82, 101
Groth, Paul 203, 278
Grove, Michael 82
- Halaschek-Wiener, Christian 82
Hendler, Jim 82
Holland, David A. 171
Hunter, Jane 212
- Jaeger-Frank, Efrat 118
Jiang, Sheng 203
- Khan, Imran 212
Khandelwal, Vaibhav 19
Kifor, Tamás 28
Kloss, Guy K. 37
Kouřil, Daniel 246
Křenek, Aleš 246
Kulkarni, Archis 19
- Ludäscher, Bertram 133
- Mann, Robert G. 193
Marru, Suresh 222
Matyska, Luděk 246
McPhillips, Timothy 133
Miles, Simon 184, 203, 278
Mills, Hugo 270
Moreau, Luc 55, 203
Mulač, Miloš 246
Muniswamy-Reddy, Kiran-Kumar 171
Munroe, Steve 203, 278
Myers, James D. 73
- Naughton, Jeffrey F. 237
- Parsia, Bijan 82
Plale, Beth 46, 222
Pospíšil, Jan 246
Prina-Ricotti, Diego 193
- Rajbhandari, Shrija 109
Rana, Omer 109
Ratnakar, Varun 90
Reilly, Christine F. 237
Ruda, Miroslav 246
- Salvet, Zdeněk 246
Santos, Emanuele 10
Schain, Andrew 82
Scheidegger, Carlos E. 10
Schreiber, Andreas 37
Schroeter, Ronald 212
Schuchardt, Karen L. 73
Seltzer, Margo I. 171
Silva, Cláudio T. 10
Simmhan, Yogesh L. 222

- Sitera, Jiří 246
Stephan, Eric G. 73
Stevens, Robert 254
- Talbott, Tara D. 73
Tan, Victor 203
Taylor, Kieron 270
Totala, Sanket 19
Tsasakou, Sofia 203
- Vansummeren, Stijn 162
Varga, László Z. 28
- Vázquez-Salceda, Javier 28
Vijayakumar, Nithya N. 46
Vo, Huy T. 10
Voců, Michal 246
- Wilde, Michael 148
Willmott, Steven 28
Wootten, Ian 109
- Zaluska, Ed 270
Zhao, Jun 254
Zhao, Yong 148