DIGITALCOMMONS
@WAYNESTATE

Wayne State University

Wayne State University Dissertations

1-1-2012

Querying and managing opm-compliant scientific workflow provenance

Chunhyeok Lim
Wayne State University,

Follow this and additional works at: http://digitalcommons.wayne.edu/oa_dissertations

Part of the Computer Engineering Commons, and the Computer Sciences Commons

Recommended Citation
Lim, Chunhyeok, "Querying and managing opm-compliant scientific workflow provenance" (2012). *Wayne State University Dissertations.* Paper 382.

This Open Access Dissertation is brought to you for free and open access by DigitalCommons@WayneState. It has been accepted for inclusion in Wayne State University Dissertations by an authorized administrator of DigitalCommons@WayneState.

www.manaraa.com

QUERYING AND MANAGING OPM-COMPLIANT SCIENTIFIC WORKFLOW PROVENANCE

by

CHUNHYEOK LIM

DISSERTATION

Submitted to the Graduate School

of Wayne State University,

Detroit, Michigan

in partial fulfillment of the requirements

for the degree of

DOCTOR OF PHILOSOPHY

2011

MAJOR: COMPUTER SCIENCE

Approved by:

_____

Advisor **Date**

_____

Co-advisor

_____

_____

_____

www.manaraa.com

©COPYRIGHT BY

CHUNHYEOK LIM

2011

All Rights Reserved

www.manaraa.com

ACKNOWLEDGMENTS

I would first like to thank God for giving me the opportunity to start my Ph.D. program in 2007 and leading me to successfully finish my Ph.D. program in 2011. I also would like to express my deep and sincere gratitude to my advisors, Dr. Farshad Fotouhi and Dr. Shiyong Lu, for their constant encouragement, support, and guidance in every time of need throughout my Ph.D. program. With their passionate advising, I successfully completed my work and also had invaluable opportunities to make myself more capable. In addition, I am grateful to my Dissertation Committee members: Dr. Robert Reynolds and Dr. Zaki Malik in the Department of Computer Science and Dr. Song Jiang in the Department of Electrical and Computer Engineering for being on my committee and giving me constructive suggestions and comments on my work.

I would like to give my special thanks to Dr. Artem Chebotko in the Department of Computer Science, University of Texas-Pan American for working hard with me as a co-author in my research papers. I would like to thank Dr. Seunghan Chang who is currently working in the Korean army for giving me lots of help to early adapt to school life. I also would like to thank my academic colleagues in the Scientific Workflow Research Laboratory: Dong Ruan, Fahima Amin, and Andrey Kashlev, and also alumni Dr. Xubo Fei, Dr. Cui Lin, and Dr. Jamal Alhiyafi for their academic cooperation and close friendships.

Many thanks go to my loving family: my wife Eunju and my sons Ugyun and Hyeongyun. They have sacrificed a lot due to my study. Without their encouragement and understanding, it would have been impossible for me to complete my program. My special gratitude goes to my loving father and mother, brothers and sister, and mother-in-law for their unconditional love and endless pray. Finally, the financial support of Republic of Korea Army is gratefully acknowledged.

www.manaraa.com

TABLE OF CONTENTS

www.manaraa.com

www.manaraa.com

www.manaraa.com

www.manaraa.com

LIST OF TABLES

www.manaraa.com

LIST OF FIGURES

www.manaraa.com

www.manaraa.com

CHAPTER 1
INTRODUCTION

Today, scientific workflows have become a powerful computing paradigm for structuring and automating complex and distributed scientific processes in various data-intensive sciences, such as bioinformatics, physics, astronomy, earthquake science, and so on [1], [2], [3], [4]. A scientific workflow is a formal specification of a scientific process, which represents, streamlines, and automates the analytical and computational steps [5], [6]. Provenance, which is one kind of metadata that captures the derivation history of a data product, including the original data sources, intermediate data products, and the workflow tasks that were applied to produce a data product, has become increasingly important in scientific workflows to interpret, validate, and analyze the result of scientific computing [7], [8]. For example, Figure 1.1(a) shows an example of a scientific workflow, which is the *Load Workflow* defined in the Third Provenance Challenge [15] that checks and reads CSV files before loading, creates a database to load CSV files, loads them into tables and validates tables, and compacts a database after loading. In general, provenance can be captured by a provenance collection mechanism during the execution of a scientific workflow. The captured provenance holds data dependencies, process dependencies, causality between data and processes, and annotations. Such provenance is often represented by a provenance graph. Figure 1.1(b) shows a provenance graph produced via the execution of the *Load Workflow*. Figure 1.1(c) and (d) present a data dependency graph associated with artifact a_9 and a process dependency graph associated with process p_8 from a provenance graph generated via the execution of the *Load Workflow*, respectively.

Recently, the Open Provenance Model (OPM) has been proposed as a standard provenance model in the community to facilitate and promote provenance interoperability among existing heterogeneous systems. The OPM model allows us to characterize what caused "things" to be (i.e.,

www.manaraa.com

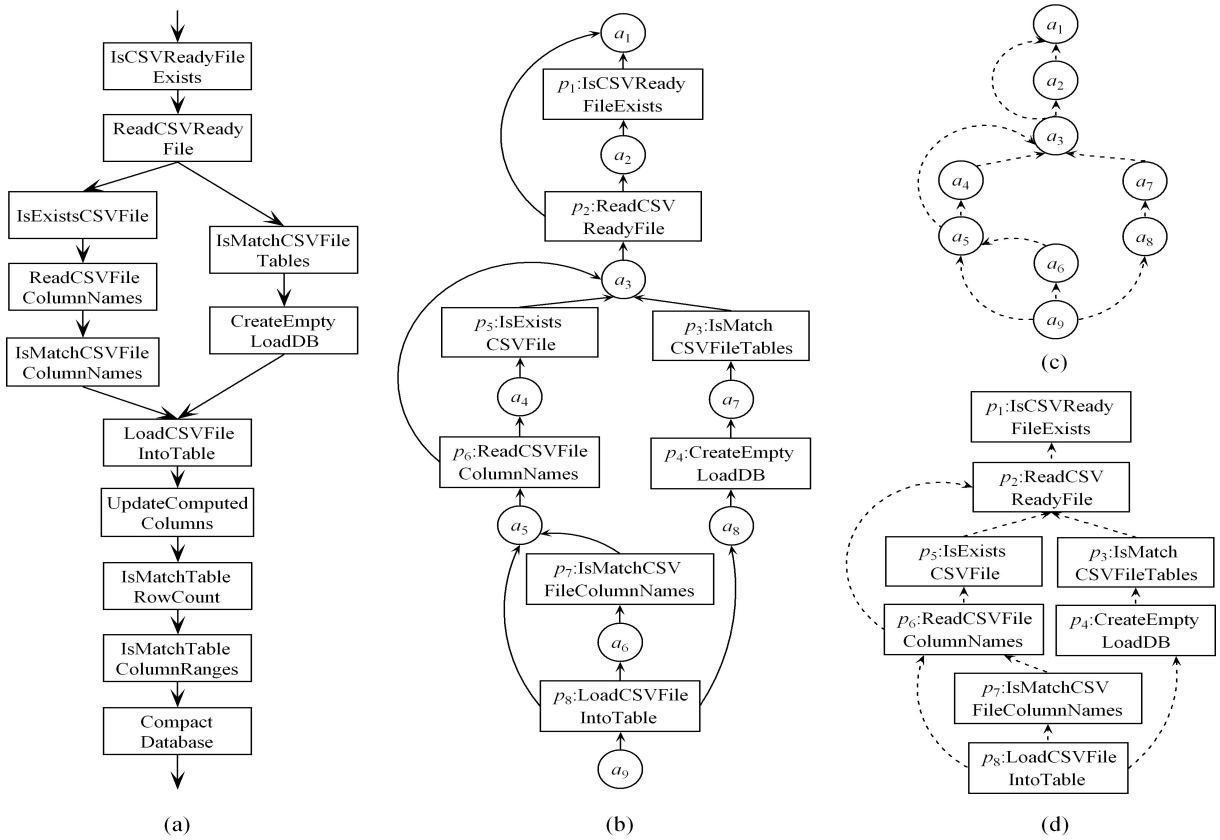


Figure 1.1: An example of a scientific workflow and its provenance.

how "things" depended on others and resulted in specific states). Therefore, the OPM model essentially consists of a directed graph to express such dependencies. We briefly introduce the constituents of such a graph. In the OPM model, provenance graphs consists of three types of nodes (i.e., *Artifact*, *Process*, *Agent*) and five types of edges (i.e., *Used*, *WasGeneratedBy*, *WasControlledBy*, *WasTriggeredBy*, *WasDerivedFrom*), which represent causal dependencies. An artifact is an immutable piece of state, a process is action or a series of actions, and an agent is a contextual entity acting as a catalyst of a process, which is enabling, facilitating, controlling, or affecting its execution. The five edges also capture the causal dependencies between the artifacts, processes, and agents. As shown in Figure 1.2, an edge represents a causal dependency between its source denoting the effect and its destination denoting the cause. The *Used* edge expresses

www.manaraa.com

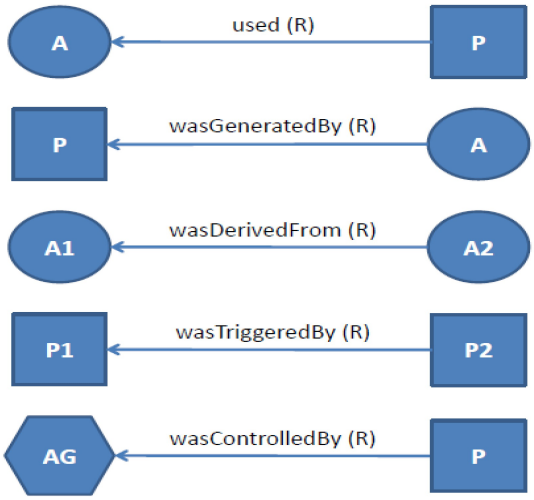


Figure 1.2: The Open Provenance Model (v1.1).

that a process used an artifact, and the *WasGeneratedBy* edge expresses that an artifact was generated by a process. The *WasControlledBy* edge also expresses that a process was controlled by an agent. Regarding edge *WasDerivedFrom*, even though an artifact A_2 may have been generated by a process that used some artifacts, this does not tell us which artifact A_2 actually depends upon. Thus, to make the dependency explicit, it is required to assert that artifact A_2 was derived from another artifact A_1. This edge gives us a dataflow oriented view of provenance. Likewise, for edge *WasTriggeredBy* it is recognized that we may not be aware of the exact artifact that a process P_2 used, but that there was some artifact generated by another process P_1. Process P_2 is then said to have been triggered by P_1. It allows for a process oriented view of past executions to be adopted. The OPM model has played an important role in provenance interoperability and has had a positive impact on ongoing provenance activities, including the IPAW workshops [11] and the Provenance Challenges [14]. More details on the OPM model can be found in [18].

There is a growing effort in supporting the OPM model in the existing systems. Most existing systems [35], [51], [62], [27], [67], [97], [90] store and manage provenance data in their own provenance stores of proprietary provenance models; however, these systems have to conduct an additional transformation procedure to store and manage OPM-compliant provenance data (i.e.,

www.manaraa.com

XML data that conforms to the XML schema defined in the OPM model) by means of a mapping between their own proprietary provenance models and the OPM model, which is cumbersome and inefficient. Moreover, most existing systems conduct query processing over the physical provenance storages (i.e., RDB, RDF, and XML) using query languages, such as SQL, SPARQL, and XQuery, which are closely coupled to the underlying provenance storage strategies; thus, users have to know the structures or schemas of such provenance storages as well as semantics of provenance models, which is nontrivial for users to formulate complicated provenance queries. Therefore, an efficient and effective provenance management mechanism is needed to query and manage OPM-compliant provenance in a native fashion.

1.1 Statement of the Problem

1.1.1 Provenance Collection Framework

In scientific workflow environments, provenance management is essential to support reproducibility of scientific discovery, result interpretation, and problem diagnosis [8], [64]. In general, provenance management concerns about the efficiency and effectiveness of recording, representing, storing, querying, and visualizing provenance. Much research has been done for provenance management [5], [35], [54], [25], [67], [90], [61], [85]. In particular, many capture mechanisms have been proposed in existing provenance systems; however, most systems capture provenance based on their own proprietary provenance models. Since the captured provenance data does not conform to the OPM XML schema when it is collected, such provenance data should be transformed via a mapping between their proprietary provenance models and the OPM model to support the OPM model. Therefore, we need a new provenance capture mechanism to directly collect OPM-compliant provenance data during the execution of a scientific workflow.

Moreover, the OPM model only models *retrospective provenance*, which captures past workflow execution and data derivation information. Another kind of provenance, called *prospective provenance*, which captures an abstract workflow specification as a recipe for future data derivation, cannot be modeled by the OPM model at this point. As a result, many provenance queries

www.manaraa.com

related to workflow specification (prospective or hybrid provenance queries) cannot be answered based on the OPM model. For example, among 16 provenance queries defined in Third Provenance Challenge [15], query OQ9, which asks for "which steps were not executed because of halt?" needs information associated with steps specified in a workflow before halt occurs in order to answer this query. However, the OPM model does not represent information associated with workflow specification existing before the execution of a workflow.

Therefore, we need to extend the OPM model to model *prospective provenance* so that we can answer many provenance queries related to workflow specification before the execution of a scientific workflow. That is, we need an efficient provenance capture mechanism that collects both prospective provenance and retrospective provenance.

1.1.2 Provenance Store for Scientific Workflows

Scientific workflows have become an increasingly popular paradigm for scientists to formalize and structure complex scientific processes to enable and accelerate many significant scientific discoveries [1], [2]. The importance of scientific workflows has been recognized by NSF since 2006 [3] and was reemphasized in a recent science article [4], which concluded, "In the future, the rapidity with which any given discipline advances is likely to depend on how well the community acquires the necessary expertise in database, *workflow management*, visualization, and cloud computing technologies." As a result, provenance management has been identified as a key component of the reference architecture for scientific workflow management systems (SWFMSs) [5]. The importance of provenance has been widely recognized in the scientific workflow community: almost all existing SWFMSs now support provenance management [5], [35], [54], [25], [67], [90], [61], [85] as a key functionality, even though challenges remain for the efficient and effective management of provenance [2].

Although numerous provenance systems [35], [51], [62], [27], [67], [97], [5], [90] have been developed to manage provenance data, either as part of a scientific workflow management system or as a standalone provenance system [61], [85], provenance interoperability is poor among these

www.manaraa.com

```
<?xml version="1.0" encoding="UTF-8" ?>
<Trace runId="a1fd241f-305e-4eff-a2d9-f6f988557980" traceId="1">
    <Data id="2661" type="StringToken">csv899251008506</value>
    <Data id="1776" type="StringToken">success-J062941-20081115-P2FrameMeta.csv</value>
    <Data id="2189" type="StringToken">J062941-success-442745064-LoadDB</value>
    <Insertion item="1712" invocation="IsExistsCSVFile:1" />
    <Insertion item="3138" invocation="ReadCSVReadyFile:1" />
    <Insertion item="2045" invocation="LoadCSVFileIntoTable:1" />
    <InvocationDependency from="IsMatchCSVFileColumn:1" to="ReadCSVFileColumn:1"/>
    <InvocationDependency from="UpdateComputedColumns:1" to="LoadCSVFileIntoTable:1"/>
    <InvocationDependency from="ReadCSVFileColumnNames:1" to="IsExistsCSVFile:1"/>
    <  ...  />
</Trace>
```

Figure 1.3: An example of the XML provenance data produced by the UCDGC team.

systems due to the use of their own proprietary provenance models [2]. Such a lack of provenance interoperability makes it difficult to integrate provenance from various heterogeneous SWFMSs, which is necessary when scientific results were obtained by running a sequence of scientific workflows enacted from different SWFMSs [80]. To address this issue, the Open Provenance Model (OPM) [12] initiative was formed in 2007 with the aim of defining a standard provenance model to facilitate provenance interoperability between different heterogeneous systems.

While an increasing number of systems have started to support the OPM model [15], most of them use an import/export approach, which extends their own proprietary provenance models with an import/export facility to map back and forth between the OPM model and their own provenance models. For example, Figure 1.3 shows an example of the XML provenance data produced by the UCDGC (UC Davis Genome Center) team in the Third Provenance Challenge [15]. As depicted in Figure 1.3, the UCDGC's XML provenance data does not conform to the XML schema defined in the OPM model [12]. Therefore, in case provenance data produced by heterogeneous provenance systems is exchanged each other, the UCDGC's provenance data should be transformed to separate provenance data that conforms to the OPM XML schema when it is exported. Figure 1.4 shows an example of XML provenance data that conforms to the OPM XML schema. In fact, the provenance data depicted in Figure 1.3 can be transformed to the provenance data presented in Figure 1.4

www.manaraa.com

```
<?xml version="1.0" encoding="UTF-8" ?>
<opmGraph xmlns="http://openprovenance.org/model/v1.01.a">
    <processes>
        <process id="IsExistsCSVFile:1">
            <value>IsExistsCSVFile</value>
        </process>
        <process id="ReadCSVReadyFile:1">
            <value>ReadCSVReadyFile</value>
        </process>
    </processes>
    <artifacts>
        <artifact id="1776">
            <value>success-P2-J062941-B001-P2fits0-20081115-P2FrameMeta.csv</value>
        </artifact>
        <artifact id="2189">
            <value>J062941-success-442745064-LoadDB</value>
        </artifact>
    </artifacts>
    <causalDependencies>
        <used>
            <effect id="IsExistsCSVFile:1" />
            <role value="in" />
            <cause id="2661" />
        </used>
        <wasGeneratedBy>
            <effect id="1776" />
            <role value="out" />
            <cause id="ReadCSVReadyFile:1" />
        </wasGeneratedBy>
        <  ...  />
    </causalDependencies>
</opmGraph>
```

Figure 1.4: An example of XML provenance data that conforms to the OPM XML schema.

via a mapping procedure. Likewise, when the OPM-compliant provenance data is imported, it should be transformed to separate provenance data that meets the UCDGC proprietary provenance model. These mapping strategies are expensive and inefficient. Therefore, we need an efficient and effective provenance management mechanism to store and manage OPM-compliant provenance data in a native fashion without any transformation procedure.

www.manaraa.com

1.1.3 Provenance Query Language

Most existing systems [61], [25], [28], [54] store provenance data in their provenance stores of proprietary provenance models and conduct provenance querying using query languages, such as SQL, SPARQL, and XQuery over the physical provenance storages (i.e., RDB, RDF, and XML). Such query languages are closely coupled to the underlying provenance storage strategies, and therefore users have to know the structures or schemas of such provenance storages, as well as semantics of provenance models that have been applied to the provenance storages to formulate provenance queries. Moreover, users require the expertise about grammars, syntax, and semantics of such languages to formulate complicated provenance queries.

For example, using existing approaches, *provenance lineage queries* (queries for tracking ancestor nodes) often require users to write recursive queries (directly typing recursive statements or using recursive functionality). Figure 1.5 shows an example of different query languages (i.e., SQL, SPARQL, and XQuery) that answer a provenance query (CQ1), which is one out of 16 provenance queries defined in the Third Provenance Challenge [15]. Query CQ1, which asks for CSV

```
CQ1: For a given detection (id), which CSV files contributed to it?

<SQL>: SELECT DISTINCT A2.Value FROM Artifact A2, Used U,
    (SELECT DISTINCT TG.OPMGraphId, TG.ProcessId FROM MultiStepWasGeneratedBy TG, Artifact A
    WHERE TG.ArtifactId = A.ArtifactId AND A.Value = 'detectID') As Pv
    WHERE U.ProcessId = Pv.ProcessId AND U.ArtifactId = A2.ArtifactId AND
    A2.Value LIKE '%Detection.csv'

<SPARQL>: SELECT ?value WHERE {?wgb ProtoProv:wgbSource pc:DBEntryP2Detection_0_ForIter3.
    ?wgb ProtoProv:wgbSource pc:DBEntryP2Detection_0_ForIter3. ?wgb ProtoProv:wgbTarget ?fxn.
    ?usd ProtoProv:usdSource ?fxn. ?usd ProtoProv:usdTarget ?var. ?var ProtoProv:hasType ?type.
    FILTER(?type = "CSVFileEntry") ?var ProtoProv:hasValue ?value}

<XQuery>: LET $d := doc('workflow_opm2.xml');
    (: the user must first find the detection value in the DB. That table is used to find the artifact :)
    LET $a := $d//artifact[value/function/parameter/@val = 'P2Detection'];
    (: return all the artifacts upstream containing a P2Detection.csv file :)
    RETURN local:derivedFrom($d, $a)[ends-with(value/function/parameter/@val, 'P2Detection.csv')]
```

Figure 1.5: An example of different query languages answering a provenance query (CQ1).

www.manaraa.com

files that contributed to a given detection, requires the computation of transitive relationships. As depicted in Figure 1.5, to answer this query (i.e., to find all data product that contributed to derive a data product via the calculation of transitive relationships), SQL uses relation *MultiStepWasGeneratedBy* predefined via recursive queries, SPARQL uses many join conditions, and XQuery uses a predefined recursive functionality, respectively. These languages require that users directly formulate provenance queries against physical provenance storages; as well as users need to understand and consider the underlying provenance storage strategies, which are nontrivial.

Therefore, we need a new provenance query language that efficiently supports provenance queries. We aim at designing *OPQL*, which is an OPM-level provenance query language. *OPQL* is a graph query language that is directly defined over the OPM model [18], which is a standard provenance model. An *OPQL* query takes one OPM graph as input and produces an OPM graph as output; therefore, *OPQL* queries are not tightly coupled to the underlying storage strategies.

1.2 Main Contributions

In this dissertation, we first propose a provenance collection mechanism that captures both prospective provenance and retrospective provenance in scientific workflow environments. The main contributions of this work are followings:

1. We design a provenance model that models both prospective provenance, which captures an abstract workflow specification as a recipe for future data derivation and retrospective provenance, which captures past workflow execution and data derivation. Our proposed provenance model is an extension to the Open Provenance Model (OPM), which only models retrospective provenance.

2. We propose a provenance collection framework to collect both prospective and retrospective provenance according to our model. It is important to model and capture both prospective and retrospective provenance since both provenance provide important contextual information for the comprehensive analysis of scientific results. In fact, two queries out of 16 queries raised in the Third Provenance Challenge cannot be answered solely based on the OPM

www.manaraa.com

model. Many provenance queries related to workflow specification can be answered via our provenance collection framework that models and captures both prospective and retrospective provenance.

Then, we propose a relational database-based provenance system, called OPMPROV that stores, reasons, and queries prospective provenance and retrospective provenance, which is OPM-compliant provenance. The main contributions of this work are followings:

1. We propose a relational provenance store to store, reason, and query prospective and retrospective provenance, which is captured via the proposed provenance collection framework. An experimental study is performed to show the performance of our provenance store using provenance queries defined in the Third Provenance Challenge. While most existing systems use an internal proprietary provenance model and develop an import/export facility to convert between the proprietary model and the OPM model, our provenance store features the native support of the OPM model.

2. We show that provenance reasoning defined in the OPM model can be sufficiently supported by OPMPROV using recursive views and SQL queries alone without any additional reasoning engine. Experiments are conducted to evaluate the performance of OPMPROV in data insertion and provenance querying and the experiment results show to be very efficient. A case study is performed, demonstrating that OPMPROV can answer all except one query out of the 16 queries defined in the Third Provenance Challenge.

Finally, we propose *OPQL*, an OPM-level provenance query language, that is directly defined over the Open Provenance Model (OPM). An *OPQL* query takes an OPM graph as input and produces an OPM graph as output. Therefore, *OPQL* queries are not tightly coupled to the underlying provenance storage strategies. The main contributions of this work are followings:

1. To design *OPQL* that efficiently supports provenance queries, we first define six types of graph patterns, which are the main building blocks of an *OPQL* query. We then define an

www.manaraa.com

OPM-based graph algebra based on four operators (i.e., extract, union, intersection, and difference operator). We finally define *OPQL* syntax and semantics that is required to formulate *OPQL* queries. Our *OPQL* features the native support for query processing of OPM graphs.

2. We implement *OPQL* using a Web service via our OPMPROV system; therefore, users can invoke the Web service to execute *OPQL* queries in a provenance browser, called OPMPROVIS. The result of *OPQL* queries is displayed as an OPM graph in OPMPROVIS. An experimental study is conducted to evaluate the feasibility and performance of OPMPROV on *OPQL* provenance querying and the experiment results show satisfactory performance. To our best knowledge, *OPQL* is the first OPM-level query language and OPM-compliant provenance querying service for scientific workflows.

1.3 Organization

The rest of this dissertation is organized as follows: Chapter 2 presents related work on provenance management in existing provenance systems. Chapter 3 presents a provenance collection framework that collects both prospective provenance and retrospective provenance. Chapter 4 presents a relational database-based provenance system, called OPMPROV that stores, reasons, and queries prospective provenance and retrospective provenance, which is OPM-compliant provenance. Chapter 5 presents *OPQL*, an OPM-level provenance query language, that efficiently supports provenance queries. Chapter 6 presents the design and implementation of OPMPROV. Finally, Chapter 7 concludes this dissertation and provides the directions for future work.

www.manaraa.com

CHAPTER 2

RELATED WORK

In this chapter, we first discuss how provenance has been used in the various domains, such as databases, Web, and scientific workflows to give a better understanding of provenance. We then discuss related work on scientific workflow provenance management in existing systems. We finally discuss our research in the context of OPM-compliant provenance management.

2.1 The Use of Provenance in Various Domains

In e-science environments, provenance has become increasingly important to trace, validate, and analyze the orgins and derivation of data. In common sense, provenance refers to the fact of coming from some particular sources; orgin; derivation. That is, provenance is a historical metadata that provides explanations on how a particular result has been generated. Accordingly, the notion of provenance has been used in the various domains, such as databases, Web, and scientific workflows [64].

In the context of scientific workflow, provenance is one kind of metadata that captures the derivation of history of a data product, including the original sources, intermediate data product, and the workflow tasks that were applied to produce a data product. Typically, scientific workflow provenance contains information about data dependencies, process dependencies, causality between data and processes, and annotations and it plays a key role in scientific workflows to trace the experiment results back to the origin, reproduce the data products, and verify a series of process that were used to produce the results.

In the field of databases, provenance, called *data provenance* refers to the process of tracing and recording the orgins of data and its movement between databases [121], [120]. The issue of *data provenance* is important in scientific databases to verify the accuracy and quality of data. In

www.manaraa.com

databases, much research on the management of *data provenance* has been done on two perspectives: one is "why" provenance, which refers to the source data that had some influence on the existence of the data and the other one is "where" provenance, which refers to the location in the source databases from which the data was extracted [119].

Similarly, in the field of Web data, provenance, called *Web data provenance* is important to evaluate qualities (i.e., accuracy, timelines, reliability, and trustworthiness) of the data retrieved from the Web [122], [123]. *Web data provenance* includes the access of data items on the Web, which is not required in the context of self-contained systems such as DBMSs or scientific workflow management systems. In this dissertation, we focus on the management of scientific workflow provenance, especially on OPM-compliant provenance management.

2.2 Storing and Querying Scientific Workflow Provenance

Scientific workflows have emerged for scientists to efficiently arrange and organize the complex scientific processes and facilitate many scientific discoveries. A scientific workflow management system is a system that supports the workflow specification, workflow scheduling, workflow execution, workflow monitoring, provenance management, and data product management. In general, provenance management concerns about the efficiency and effectiveness of recoding, storing, representing, querying, and visualizing provenance data. Much research on scientific workflow provenance management has been done in existing systems.

Kepler [35], [36] implements a provenance framework, called COMAD (*Collection-Oriented Modeling and Design*), which supports nested data collections and captures explicit data dependencies. The COMAD framework stores provenance information (trace) in an XML file by means of a set of provenance annotations. Recently, the COMAD framework has been extended to automatically store provenance information in a relational database, where immediate and transitive closure dependencies derived from provenance reasoning for each node and invocation are stored

www.manaraa.com

by applying a set of reduction techniques to reduce the storage cost [33], [34]. The COMAD-Kepler provenance system supports provenance querying through a high-level query language, called QLP and an external reasoning engine.

Taverna [51], [56] implements a logbook plugin to capture provenance information from workflow runs based on a provenance ontology. The logbook plugin allows users to browse, reload, rerun, and maintain provenance metadata. Taverna presents a data lineage model to support fine-grained and efficient lineage querying of collection-based workflow provenance [50]. Taverna uses Semantic Web technologies for representing provenance metadata and a general-purpose RDF store to manage and query provenance [54]. Recently, Taverna [54] implements a semantic provenance infrastructure and visualizes semantic, RDF-based provenance graphs based on a provenance ontology. Taverna supports provenance queries using the SPARQL query language.

Karma [61] captures uniform and usable provenance metadata independent of the used workflow or service framework. The Karma provenance model captures two forms of provenance: *process provenance*, which is metadata describing workflow execution and associated invocations; and *data provenance*, which provides similar metadata about the derivation history of a data product. Karma's provenance model consists of two levels: the *registry* level, which records the metadata of services and data that may be used in an execution sequence; and the *execution* level, which models instances of the registry level and records the execution-related information of method invocations and data products used or generated by each invocation [60]. Karma uses XML and relational database technologies to store and query provenance metadata [60], [63], [64]. Recently, Karma [61] presents an integrated provenance management architecture that supports automated data provenance collection, annotated provenance, and provenance visualization. Karma supports provenance queries in SQL and XPath.

VisTrails [27], [26] is the first one to support provenance tracking of workflow evolution. In VisTrails, workflow evolution provenance is represented as a version tree, in which each node corresponds to a version of a workflow, and each edge corresponds to an update action that was applied to the parent workflow to create the child workflow. VisTrails uses XML and relational

www.manaraa.com

database technologies for provenance management. VisTrails [25] uses a change-based provenance mechanism to capture provenance information for data products and for the evolution of the workflows used to generate these products. The provenance model consists of three layers: the *workflow evolution* layer, which captures the evolution relationship between workflow specifications; the *workflow* layer, which consists of individual workflow specifications; and the *execution* layer, which stores run-time information of workflow execution. VisTrails has the ability to visualize query results by highlighting workflow versions that match query conditions by using the VisTrails query language, called vtPQL.

Swift [67] is a scientific workflow management system that has focused on the rapid and reliable specification, execution, and management of large-scale science and engineering workflows. Swift implements a Virtual Data System (VDS) consisting of a set of relations to store the description of executable programs as transformations, their actual invocations as derivations, and input/outputs as data objects. Swift uses provenance for tracking the data derivation history, on-demand data generation, and data product validation. Swift utilizes relational database technologies to manage and query provenance metadata.

PReServ/PASOA [97] supports the recording of interaction provenance, actor provenance, and input provenance with the provenance recording protocol, which specifies the messages that actors can asynchronously exchange with a provenance store to support provenance submission. PReServ [84] uses a provenance management service that provides a common interface to enable different storage systems, such as file systems, relational databases, XML databases, and RDF stores, as a provenance store.

The summary of storage and query capabilities, including provenance capture mechanisms for above described systems is shown in Table 2.1. These systems have shown their storage and querying capabilities on a sample scientific workflow defined in the Third Provenance Challenge [15].

www.manaraa.com

Table 2.1: The characteristics of provenance management systems.

	Kepler	***Taverna***	***Karma***	***Vistrails***	***Swift***	***PreServ***
Scientific Domain	Biology Ecology Geology	Biology	Biology	Ecology Meterology	Biology	Biology
Capture Mechanism	Application-Oriented (COMAD)	Application-Oriented (Plug-in)	Service-Oriented	Application-Oriented (Change-based)	Application-Oriented (Plug-in)	Service-Oriented (PReP)
Representation	XML	XML/RDF	XML	XML	XML	XML
Storage	RDBMS	RDF Store	XML Database	RDBMS	RDBMS	RDBMS
Query Language	QLP	SPARQL	SQL/XPath	vtPQL	SQL	SQL

2.3 Querying and Managing OPM-Compliant Provenance

In scientific workflow environments, provenance management has become an essential functionality for most scientific workflow management systems. In 2006, the issue of provenance interoperability was first raised an important part of the provenance management and it has been actively discussed in the community [14]. To promote and facilitate interoperability among heterogeneous provenance systems, the Open Provenance Model (OPM) [17] was first proposed in 2008 and afterwards has played an important role in community activities, including the IPAW workshops and Provenance Challenges. Recently, there has been an increasing effort in adapting existing provenance systems to support OPM in the Third Provenance Challenge [15].

First, we discuss related work on adaptability to the OPM model in existing provenance systems. Kepler shows the import/export capability for the OPM model in the Third Provenance Challenge by exporting COMAD-Kepler provenance traces into OPM traces and importing the OPM traces back into the COMAD-Kepler provenance traces and then storing them into a relational database to query the imported provenance metadata. Taverna exports OPM-compliant provenance metadata from its native provenance system by means of incorporating OPM graph generation functionality into the existing provenance query algorithm for a mapping of Taverna's proprietary model to the OPM model. Karma shows that three entities and five causal dependencies defined in the OPM model can be represented as the corresponding entities in Karma by means of

www.manaraa.com

mapping the OPM model to the proprietary model. While Karma exports OPM-compliant provenance metadata from its proprietary storage, it lacks the import function and inference support for multi-step edges defined in the OPM model. In the Third Provenance Challenge, VisTrails exports OPM-compliant provenance metadata by combining information from the execution log with the workflow specification and the module registry. VisTrails uses XQuery to query the XML specifications exported and implements recursive functions to query the transitive closure dependencies. Swift is showcased to export OPM graphs from its proprietary RDBMS-based storage, however similarly to Karma, Swift has no support for importing OPM-compliant provenance or multi-step inferences. PReServ exports OPM-compliant provenance metadata by using a translation tool, and PReServ also exposes provenance graphs with three different level of abstraction, such as dependency level, process level, and communication level to describe the exported provenance metadata. Although above described systems have storage and query capabilities to adapt the OPM model in their systems, these systems have focused on enhancing the import/export capabilities in their systems by means of a mapping between their own proprietary provenance models and the OPM model. Our proposed provenance system (aka OPMPROV), on the other hand, starts from the OPM model and designs the database to store and query native OPM-compliant provenance data.

Second, most existing provenance management systems capture provenance data based on their own proprietary provenance models, and therefore the provenance data captured by these systems, which does not conforms to the OPM XML schema, should be transformed via a mapping procedure to support the OPM model, which is cumbersome and inefficient. The approach taken by our proposed provenance capture mechanism, however, differs from existing systems as our provenance capture mechanism directly captures OPM-compliant provenance data, which conforms to the OPM XML schema. The captured provenance data can be stored and managed in our provenance store, which directly uses the OPM model as the native model, without any transformation. Moreover, the OPM model only models retrospective provenance, which captures past workflow execution and data derivation information. Therefore, we extends the OPM model to support the modeling of prospective provenance, which captures an abstract workflow specification as a recipe

www.manaraa.com

for future data derivation so that many provenance queries related by workflow specification can be answered based on the OPM model.

Finally, we discuss related work on provenance query processing in provenance management systems. Most existing systems store provenance data in their provenance stores of proprietary provenance models and conduct provenance querying using query languages, such as SQL, SPARQL, and XQuery over the physical provenance storages (i.e., RDB, RDF, and XML). VisTrails has the ability to visualize query results by highlighting workflow versions that match query conditions by using the VisTrails query language, called vtPQL. Kepler [28] implements an interactive provenance browser to visualize and query data dependency graphs. The provenance browser enables users to create different views for provenance graphs and express complex and recursive graph queries. Similar to our proposed query language (aka *OPQL*), the Kepler's QLP query language provides a separation between the logical provenance model and its underlying physical representation. However, QLP is not directly defined over the OPM model, but on Kepler's proprietary provenance model. Thus, QLP has no support for direct query processing of OPM graphs. ZOOM [44] enables users to construct appropriate user views for provenance graphs, and it provides users with an interface to query provenance information. Taverna [54] implements a semantic provenance infrastructure and visualizes semantic, RDF-based provenance graphs based on a provenance ontology. Taverna supports provenance queries using the SPARQL query language. Karma [61] presents an integrated provenance management architecture that supports automated data provenance collection, annotated provenance, and provenance visualization. The Karma's provenance browser visualizes OPM graphs by a mapping between provenance events and OPM entities. Karma supports provenance queries in SQL and XPath. GraphQL [98] is a graph-based query language for graph databases. GraphQL is defined over a data model representing attributes of a generic graph, and a GraphQL query takes a collection of graphs as input and produces a collection of graphs using graph patterns. Like SQL, SPARQL, and XQuery, GraphQL requires users to directly formulate recursive queries to track ancestor nodes. Although most existing systems have the capabilities to query provenance data in their systems, query languages supported by

www.manaraa.com

these systems are closely coupled to the underlying provenance storage strategies. Moreover, these systems query OPM graphs by means of a mapping between their proprietary provenance models and the OPM model. *OPQL*, on the other hand, is directly defined over the OPM model; therefore, *OPQL* is not tightly coupled to the underlying provenance storage strategies. Our *OPQL* features the native support for query processing of OPM graphs. That is, an *OPQL* query takes one OPM graph as input and produces an OPM graph as output. *OPQL* might be a cornerstone for a study on OPM-level provenance query languages.

www.manaraa.com

CHAPTER 3
PROVENANCE COLLECTION FRAMEWORK

In this chapter, we propose a provenance collection framework that collects both prospective provenance and retrospective provenance in scientific workflow environments.

3.1 The Problem

As the OPM model has emerged in the community to promote and facilitate provenance interoperability between different heterogeneous systems, an increasing number of systems have started to support the OPM model [15]. However, most of them use an import/export approach, which extends their own proprietary provenance models with an import/export facility to map back and forth between the OPM model and their own provenance models. Moreover, the OPM model only models *retrospective provenance*, which captures past workflow execution and data derivation information. Another kind of provenance, called *prospective provenance*, which captures an abstract workflow specification as a recipe for future data derivation, cannot be modeled by the OPM model at this point. As a result, many provenance queries related to workflow specification (prospective or hybrid provenance queries) cannot be answered based on the OPM model. For example, two queries out of the 16 queries raised in the Third Provenance Challenge cannot be answered solely based on the OPM model [15].

To address these issues, we aim at designing a new provenance capture mechanism that directly captures OPM-compliant provenance data (which conforms to the OPM XML schema) as retrospective provenance, as well as that captures prospective provenance. In particular, we design a provenance model that models both prospective and retrospective provenance as an extension to the OPM model, which only models retrospective provenance. We then present a provenance collection framework to collect both prospective and retrospective provenance according to our

www.manaraa.com

model. To our best knowledge, our proposed provenance collection framework is the first provenance capture mechanism that supports the OPM model in a native fashion.

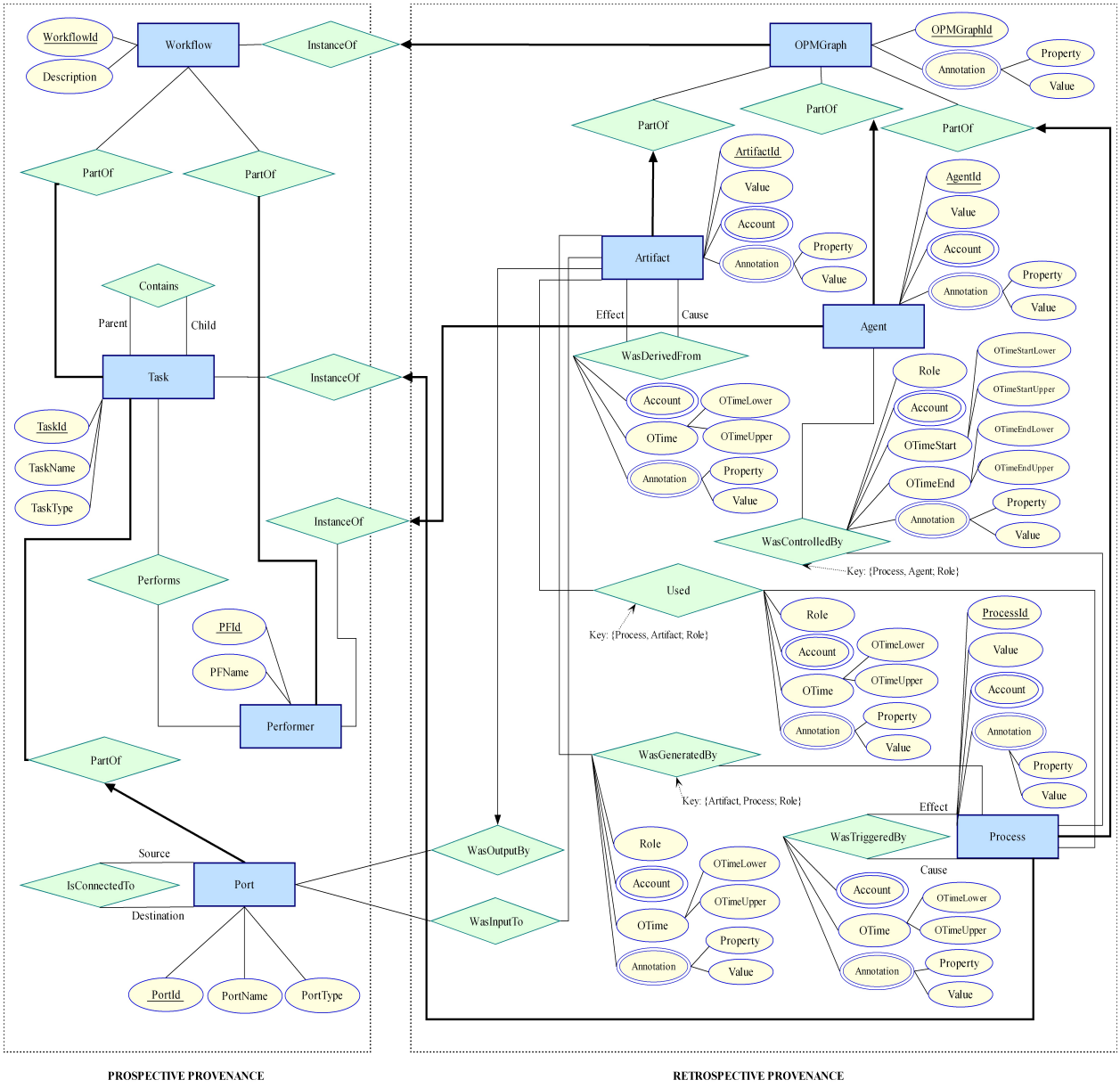


Figure 3.1: E-R diagram for modeling prospective and retrospective provenance.

www.manaraa.com

3.2 Provenance Model

In this section, we present a provenance model that deals with both prospective and retrospective provenance in scientific workflows. The model is captured via an entity-relationship diagram as shown in Figure 3.1, where the left part corresponds to the prospective provenance model and the right part corresponds to the retrospective provenance model.

3.2.1 Prospective Provenance

Prospective provenance models an abstract workflow specification as a recipe for future data derivation. Unlike a workflow specification, which can be executed by a workflow engine according to a particular scientific workflow model, prospective provenance is, in general, independent from a scientific workflow model and intended to capture the recipe in an abstract and informative form to allow further querying of this information. Prospective provenance can be automatically captured by a workbench in which a workflow design is performed. Our prospective provenance model includes four entity types, *Workflow*, *Task*, *Performer*, and *Port*, and four relationship types, *Contains*, *Performs*, *IsConnectedTo*, and *PartOf*. Each entity type has a primary key attribute that is underlined and each relationship type has a primary key that consists of the participating entity type roles. *Workflow* corresponds to a high-level notion of a workflow; it has an identifier and description. Entity type *Task* represents a computational task that is part of (relationship type *PartOf*) *Workflow*. A task can be composite, i.e., it may contain child tasks, which is captured by relationship type *Contains*. Entity type *Performer* represents a subject, such as a scientist or workflow engine, that performs (relationship type *Performs*) a task. *Performer* is also part of *Workflow*. Entity type *Port* represents an input or output port of a task; two ports can be connected to form a dataflow as captured by relationship type *IsConnectedTo*.

3.2.2 Retrospective Provenance

Retrospective provenance models past workflow execution and data derivation information, i.e., which tasks were performed and how data artifacts were derived. Retrospective provenance can be

www.manaraa.com

automatically captured during workflow execution by a workflow engine. The retrospective provenance model presented in the E-R diagram is based on the Open Provenance Model (OPM) [17] and includes four entity types, *OPMGraph*, *Process*, *Artifact*, and *Agent*, and six relationship types, *Used*, *WasGeneratedBy*, *WasControlledBy*, *WasTriggeredBy*, *WasDerivedFrom*, and *PartOf*. These entity and relationship types have direct counterparts in OPM (see [18] for their semantics), except relationship type *PartOf* which is implicit in the OPM model. According to the OPM model, graphs, artifacts, processes, and agents are identified by unique identifiers and the causal dependency edges are identified by their sources, destinations, and roles (for those that have roles) [18]. Thus, in the retrospective model, each corresponding entity type has a primary key attribute that is underlined and each relationship type has a composite primary key that includes the two roles* of the relationship with participating entity types and the *Role* attribute (for those that have the *Role* attribute). For example, the primary key of the *Used* relationship type is *ProcessId*, *ArtifactId*, and *Role* and the *WasDerivedFrom* relationship type takes *ArtifactId* for the *Effect* role and *ArtifactId* for the *Cause* role as the primary key. Each entity type has the *Value* and *Account* attributes; the latter is a set-valued attribute, such that a process, artifact, or agent can have multiple accounts. The relationship types have set-valued *Account* attributes and composite *OTime* attributes (*OTimeStart* and *OTimeEnd* for the *WasControlledBy* relationship type). Composite attribute *OTime* is composed of the *OTimeLower* and *OTimeUpper* attributes which are consistent with the *OTime* annotation in the OPM model. The *WasControlledBy* relationship type has two composite attributes *OTimeStart* and *OTimeEnd* that are composed of (*OTimeStartLower*, *OTimeStartUpper*) and (*OTimeEndLower*, *OTimeEndUpper*), respectively. Finally, each entity/relationship type has a set-valued and composite attribute *Annotation*, such that *Process*, *Artifact*, *Agent*, *Used*, *WasGeneratedBy*, *WasControlledBy*, *WasTriggeredBy*, and *WasDerivedFrom* can have multiple annotations consisting of property-value pairs defined in the OPM model.

*Note that the term "role" is used in both E-R diagram and the OPM model, but they have slightly different meanings: roles in an E-R diagram represent the participation relationships between an entity type and a relationship type, while roles in the OPM model represent annotations on *used*, *wasGeneratedBy*, and *wasControlledBy*.

www.manaraa.com

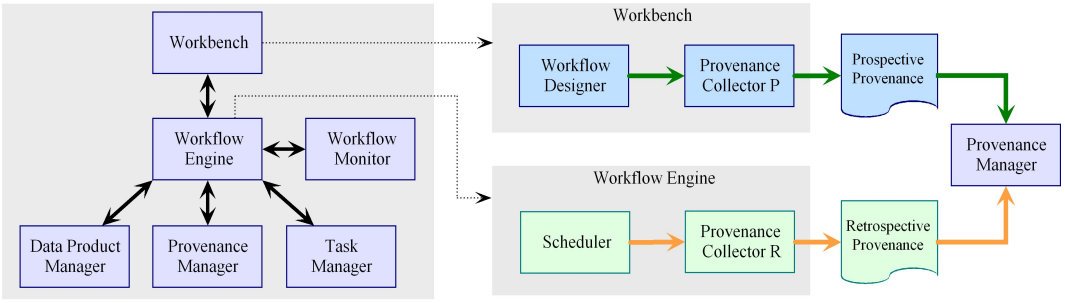
The relationship between the prospective and retrospective provenance models is captured by relationship types *InstanceOf*, *WasOutputBy*, and *WasInputTo*, such that *OPMGraph*, *Process*, and *Agent* are runtime instantiations of *Workflow*, *Task*, and *Performer*, respectively, and *Artifact* can be consumed or produced by *Port*.

3.3 Provenance Collection Framework

In this section, we propose a provenance collection framework that supports for collecting both prospective and retrospective provenance according to our provenance model. In accordance with the reference architecture for scientific workflow management systems [5], we design two provenance collectors as the core of the provenance collection framework, which are positioned in two subsystems of the reference architecture, respectively. Figure 3.2(a) depicts the system architecture for VIEW [5], which is composed of six subsystems, including workbench, workflow engine, workflow monitor, data product manager, provenance manager, and task manager. Figure 3.2(b) depicts an overview of the provenance collection framework, in which two provenance collectors are located in a workbench and workflow engine, respectively.

3.3.1 Prospective Provenance Collection

Prospective provenance captures an abstract workflow specification as a recipe for future data derivation. As shown in Figure 3.2(b), the workbench of the VIEW system features a workflow



(a) System architecture for VIEW (b) Overview of the provenance collection framework

Figure 3.2: Provenance collection framework.

www.manaraa.com

designer component that allows visual design of a workflow by a scientist. The workflow designer interacts with *Provenance Collector P*, which gathers prospective provenance. In particular, each time a workflow design specification is updated and saved, *Provenance Collector P* translates the specification into a prospective provenance document and stores it into a provenance store (aka provenance manager).

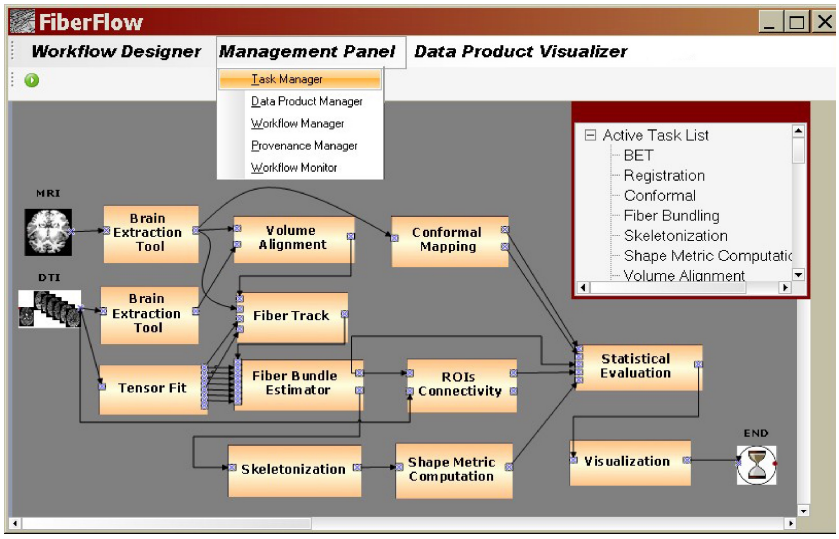


Figure 3.3: A sample workflow specification designed in the VIEW workbench.

Table 3.1: The entities and their attributes collected by *provenance collector P*.

Entity	***Attributes***
Workflow	WorkflowId, Description
Performer	PFId, PFName
Port	PortId, PortName, PortType
Contains	ParentTastId, ChildTaskId
Performs	PFId, TaskId
IsConnectedTo	SourcePortId, DestinationPortId

For example, Figure 3.3 shows a sample workflow specification in the VIEW workbench [5] and Table 3.1 shows its corresponding provenance entities and attributes collected by *Provenance Collector P* as prospective provenance.

www.manaraa.com

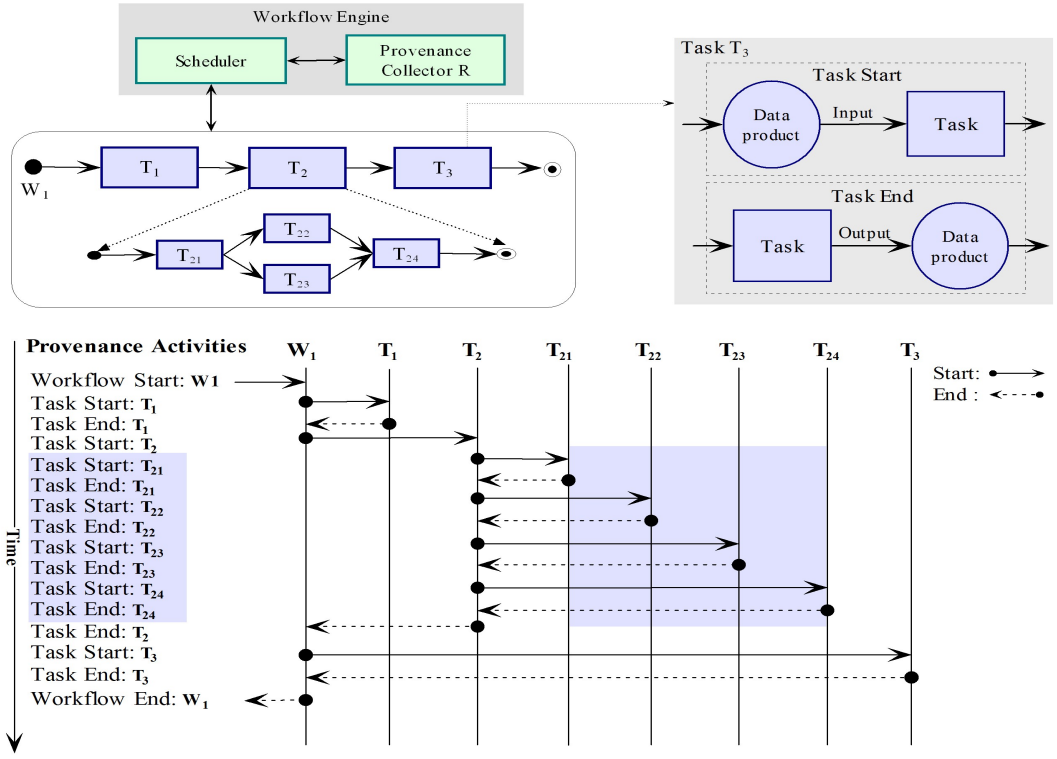


Figure 3.4: A sequence diagram illustrating provenance activities during an workflow execution.

Table 3.2: The entities and their attributes collected by *provenance collector R*.

Activity	***Entity***	***Attributes***
Workflow Start	OPMGraph	OPMGraphId
	Artifact	ArtifactId, Value, Account
	WasInputTo	ArtifactId, PortId
Workflow End	WasOutputBy	ArtifactId, PortId
Task Start	Process	ProcessId, Value, Account
	Agent	AgentId, Value, Account
	Used	ProcessId, Role, ArtifactId, Account, Timestamp
	WasControlledBy	ProcessId, Role, AgentId, Account, Timestamp
	WasInputTo	ArtifactId, PortId
Task End	Artifact	ArtifactId, Value, Account
	WasGeneratedBy	ArtifactId, Role, ProcessId, Account, Timestamp
	WasDerivedFrom	EffectArtifactId, CauseArtifactId, Account, Timestamp
	WasOutputBy	ArtifactId, PortId

www.manaraa.com

3.3.2 Retrospective Provenance Collection

Retrospective provenance captures past execution and data derivation information. As depicted in Figure 3.2(b), the workflow engine of the VIEW system uses a scheduler to schedule and execute a workflow specification obtained from the workflow designer. The workflow scheduler communicates with *Provenance Collector R*, which gathers retrospective provenance. As shown in Figure 3.4, retrospective provenance is captured for each provenance-aware activity that is scheduled by the scheduler. The sequence diagram in Figure 3.4 describes how such activities span in time during a sample workflow execution. Four types of provenance-aware activities are defined: *Workflow Start*, *Workflow End*, *Task Start*, and *Task End*. For each type of activity, captured provenance information is shown in Table 3.2. For the *Workflow Start* activity, the provenance collector collects entities *OPMGraph*, *Artifact*, and *WasInputTo* and their corresponding attributes as depicted in Algorithm 1. When the *Workflow End* activity is executed, entity *WasOutputBy* is only collected to avoid redundant information according to Algorithm 2. The *Task Start* activity collects provenance information about *Process*, *Agent*, *Used*, *WasControlledBy*, and *WasInputTo* as shown in Algorithm 3. The *Task End* activity collects provenance information of *Artifact*, *WasGeneratedBy*, *WasDerivedFrom*, and *WasOutputBy* as presented in Algorithm 4. *Provenance Collector R* stores a retrospective provenance document into the provenance store once all activities are completed. The rest of retrospective provenance information (see Table 3.3) defined in the model is not directly gathered by the provenance collector, but rather inferred using reasoning techniques.

Table 3.3: The entities and their attributes inferred by provenance reasoning.

Non-Activity	***Entity***	***Attributes***
Provenance Reasoning	WasTriggeredBy	EffectProcessId, CauseProcessId, Account, Timestamp
	Used*	ProcessId, ArtifactId, Account
	WasGeneratedBy*	ArtifactId, ProcessId, Account
	WasDerivedFrom*	EffectArtifactId, CauseArtifactId, Account
	WasTriggeredBy*	EffectProcessId, CauseProcessId, Account

www.manaraa.com

Algorithm 1 Algorithm for collecting OPM entities during activity *Workflow Start*

1: **function:** *collectOPMEntityByWorkflowStart*
2: **input:** Workflow identifier wid and workflow input list $<ptid, dpid, dpvl, dp^{acc}>$, where $ptid$ is a port identifier, $dpid$ is a data product identifier, $dpvl$ is a value of $dpid$, and dp^{acc} is an account of $dpid$
3: **output:** XML document ($xmldoc$) recording entities *OPMGraph*, *Artifact*, and *WasInputTo*
4: xmldoc = new xmldocument();
5: root_elem = xmldoc.createElement("opmGraph");
6: artifacts_elem = xmldoc.createElement("artifacts");
7: dataChannels_elem = xmldoc.createElement("dataChannels");
8: root_elem.append(artifacts_elem);
9: root_elem.append(dataChannels_elem);
10: xmldoc.append(root_elem);
11: **let** ϕ be a function to return a workflow run identifier ($wrid$) corresponding to a workflow identifier (wid), defined by $wrid = \phi(wid)$;
12: **let** ω_a be a function to return an artifact identifier (aid) corresponding to a data product ($dpid$), defined by $aid = \omega_a(dpid)$;
// recording entity "OPMGraph"
13: $opmgraphId = \phi(wid)$;
14: root_elem.setAttribute("id", $opmgraphId$);
// recording entities "WasInputTo" and "Artifact"
15: **for** each wi in worflow input list **do**
16: $portId = wi.ptid$;
17: $artifactId = \omega_a(wi.dpid)$;
18: $value = wi.dpvl$;
19: $account = wi.dp^{acc}$;
// part of entity "WasInputTo"
20: wasInputTo_elem = xmldoc.createElement("wasInputTo");
21: port_elem = xmldoc.createElement("port");
22: port_elem.setAttribute("id', $portId$);
23: input_artifact_elem = xmldoc.createElement("inputArtifact");
24: input_artifact_elem.setAttribute("id", $artifactId$);
25: wasInputTo_elem.append(port_elem);
26: wasInputTo_elem.append(input_artifact_elem);
27: dataChannels_elem.append(wasInputTo_elem);
// part of entity "Artifact"
28: artifact_elem = xmldoc.createElement("artifact");
29: artifact_elem.setAttribute("id", $artifactId$);
30: account_elem = xmldoc.createElement("account");
31: account_elem.setAttribute("id", $account$);
32: value_elem = xmldoc.createElement("value");
33: value_elem.setAttribute("id", $value$);
34: artifact_elem.append(account_elem);
35: artifact_elem.append(value_elem);
36: artifacts_elem.append(artifact_elem);
37: **end for**
38: **return** $xmldoc$;
39: **end function**

www.manaraa.com

Algorithm 2 Algorithm for collecting OPM entities during activity *Workflow End*

```
1:  function: collectOPMEntityByWorkflowEnd
2:  input: Workflow output list <ptid, dpid>, where ptid is a port identifier and dpid is a data product identifier
3:  output: XML document (xmldoc) recording entity WasOutputBy
4:  xmldoc = new xmldocument();
5:  root_elem = xmldoc.createElement("opmGraph");
6:  dataChannels_elem = xmldoc.createElement("dataChannels");
7:  root_elem.append(dataChannels_elem);
8:  xmldoc.append(root_elem);
9:  let ω_a be a function to return an artifact identifier (aid) corresponding to a data product (dpid), defined by aid = ω_a(dpid);
    // recording entity "WasOutputBy"
10: for each wo in worflow output list do
11:     portId = wo.ptid;
12:     artifactId = ω_a(wo.dpid);
13:     wasOutputBy_elem = xmldoc.createElement("wasOutputBy");
14:     port_elem = xmldoc.createElement("port");
15:     port_elem.setAttribute("id", portId);
16:     output_artifact_elem = xmldoc.createElement("outputArtifact");
17:     output_artifact_elem.setAttribute("id", artifactId);
18:     wasOutputBy_elem.append(port_elem);
19:     WasOutputBy_elem.append(output_artifact_elem);
20:     dataChannels_elem.append(wasOutputBy_elem);
21: end for
22: return xmldoc;
23: end function
```

3.4 Summary

In this chapter, we designed a provenance model that models both prospective provenance, which captures an abstract workflow specification as a recipe for future data derivation and retrospective provenance, which captures past workflow execution and data derivation. Then, we proposed a provenance collection framework to capture both prospective and retrospective provenance. Our provenance collection framework features the native support for the OPM model.

www.manaraa.com

Algorithm 3 Algorithm for collecting OPM entities during activity *Task Start*

function: *collectOPMEntityByTaskStart*
input: Task identifier tid, task name $tname$, task account t^{acc}, performer identifier $pfid$, performer name $pfname$, performer account pf^{acc} and task input list $<ptid, dpid, dpvl, dp^{acc}>$, where $ptid$ is a port identifier, $dpid$ is a data product identifier, $dpvl$ is a value of $dpid$, and dp^{acc} is an account of $dpid$
output: XML document ($xmldoc$) recording entities *Process*, *Agent*, *Used*, *WasControlledBy*, and *WasInputTo*
xmldoc = new xmldocument();
root_elem = xmldoc.createElement("opmGraph");
processes_elem = xmldoc.createElement("processes");
agents_elem = xmldoc.createElement("agents");
dependencies_elem = xmldoc.createElement("causalDependencies");
dataChannels_elem = xmldoc.createElement("dataChannels");
root_elem.append(processes_elem);
root_elem.append(agents_elem);
root_elem.append(dependencies_elem);
root_elem.append(dataChannels_elem);
xmldoc.append(root_elem);
let ω_p and ω_{ag} be functions to return a process identifier (pid) and agent identifier ($agid$) corresponding to a task ($dpid$) and performer ($pfid$), repectively, defined by $pid = \omega_p(tid)$ and $agid = \omega_{ag}(pfid)$;
$processId = \omega_p(tid)$;
$agentId = \omega_{ag}(pfid)$;
$p\_value = tname$;
$ag\_value = pfname$;
$p\_account = t^{acc}$;
$ag\_account = pf^{acc}$;
// recording entity "Process"
process_elem = xmldoc.createElement("artifact");
process_elem.setAttribute("id", $processId$);
paccount_elem = xmldoc.createElement("account");
paccount_elem.setAttribute("id", $p\_account$);
pvalue_elem = xmldoc.createElement("value");
pvalue_elem.setAttribute("id", p_value);
process_elem.append(paccount_elem);
process_elem.append(pvalue_elem);
processes_elem.append(process_elem);
// record the "Agent" entity in a similar fashion
// recording entities "Used", "WasControlledBy", and "WasInputTo"
for each ti in task input list **do**
    $portId = ti.ptid$;
    $artifactId = \omega_a(ti.dpid)$;
    $a\_value = ti.dpvl$;
    $a\_account = ti.dp^{acc}$;
    $effectId = processId$;
    $role$ = "$used$";
    $causeId = artifactId$;
    $used\_account = p\_account$;
// part of entity "Used"
    used_elem = xmldoc.createElement("used");
    effect_elem = xmldoc.createElement("effect");
    effect_elem.setAttribute("id", $effectId$);
    role_elem = xmldoc.createElement("role");
    role_elem.setAttribute("id", $role$);
    cause_elem = xmldoc.createElement("cause");
    cause_elem.setAttribute("id", $causeId$);
    account_elem = xmldoc.createElement("account");
    account_elem.setAttribute("id", $used\_account$);
    used_elem.append(effect_elem);
    used_elem.append(role_elem);
    used_elem.append(cause_elem);
    used_elem.append(account_elem);
    dependencies_elem.append(used_elem);
// In a similar fashion, record entities "WasControlledBy" and "WasInputTo"
end for
return $xmldoc$;
end function

www.manaraa.com

Algorithm 4 Algorithm for collecting OPM entities during activity *Task End*

function: *collectOPMEntityByTaskEnd*
input: Task identifier tid, task account t^{acc}, and task output list $<ptid, dpid, dpvl, dp^{acc}>$, where $ptid$ is a port identifier, $dpid$ is a data product identifier, $dpvl$ is a value of $dpid$, and dp^{acc} is an account of $dpid$
output: XML document ($xmldoc$) recording entities *Artifact*, *WasGeneratedBy*, *WasDerivedFrom*, and *WasOutputBy*
xmldoc = new xmldocument();
root_elem = xmldoc.createElement("opmGraph");
artifacts_elem = xmldoc.createElement("artifacts");
dependencies_elem = xmldoc.createElement("causalDependencies");
dataChannels_elem = xmldoc.createElement("dataChannels");
root_elem.append(artifacts_elem);
root_elem.append(dependencies_elem);
root_elem.append(dataChannels_elem);
xmldoc.append(root_elem);
let ω_p and ω_a be functions to return a process identifier (pid) and agent identifier (aid) corresponding to a task (tid) and data product ($dpid$), repectively, defined by $pid = \omega_p(tid)$ and $aid = \omega_a(dpid)$;
$processId = \omega_p(tid)$;
$p\_account = t^{acc}$;
// recording entities "Artifact", "WasGeneratedBy", "WasDerivedFrom", and "WasOutputBy"
for each to in task output list **do**
    $portId = to.ptid$;
    $artifactId = \omega_a(to.dpid)$;
    $a\_value = to.dpvl$;
    $a\_account = to.dp^{acc}$;
    $effectId = artifactId$;
    $role$ = "$generated$";
    $causeId = processId$;
    $generated\_account = paccount$;
// part of entity "Artifact"
    ...
    record the "Artifact" entity in a similar fashion as described in Algorithm 2
// part of entity "WasGeneratedBy"
    wasGeneratedBy_elem = xmldoc.createElement("wasGeneratedBy");
    effect_elem = xmldoc.createElement("effect");
    effect_elem.setAttribute("id", $effectId$);
    role_elem = xmldoc.createElement("role");
    role_elem.setAttribute("id", $role$);
    cause_elem = xmldoc.createElement("cause");
    cause_elem.setAttribute("id", $causeId$);
    account_elem = xmldoc.createElement("account");
    account_elem.setAttribute("id", $generated\_account$);
    wasGeneratedBy_elem.append(effect_elem);
    wasGeneratedBy_elem.append(role_elem);
    wasGeneratedBy_elem.append(cause_elem);
    wasGeneratedBy_elem.append(account_elem);
    dependencies_elem.append(wasGeneratedBy_elem);
// In a similar fashion, record entities "WasDerivedFrom" and "WasOutputBy"
end for
return $xmldoc$;
end function

www.manaraa.com

CHAPTER 4

THE OPMPROV PROVENANCE STORE

In this chapter, we propose OPMPROV, a relational database-based provenance system, that stores, reasons, queries prospective provenance and retrospective provenance, which is OPM-compliant provenance.

4.1 The Problem

As described previously, provenance is essential for scientific workflows to support reproducibility of scientific discovery, result interpretation, and problem diagnosis [8], [64]. Although numerous provenance systems [35], [51], [62], [27], [67], [97], [5], [90] have been developed, their interoperability is poor due to the lack of a common data model for provenance. To address this issue, the OPM model [12] was proposed. Since then, the OPM model has played an important role in provenance interoperability.

While there is a growing effort in supporting the OPM model in existing scientific workflow provenance systems [37], [51], [62], [27], [67], [97] and the evaluation of the OPM model in a particular domain [9], [10], such as the scientific workflow domain, most of them focus on enhancing an existing provenance system with the import/export capability for the OPM model. In this dissertation, however, we take the OPM model as a starting point and develop a native OPM provenance store. By native, we mean that the OPM model is the conceptual data model that is used to design our provenance store and the input and output of such a store is OPM-compliant provenance data. Therefore, our work complements the existing work whose OPM support is based on back and forth transformations between the OPM model and proprietary models employed by these systems.

www.manaraa.com

Although using the OPM model as an implementation schema belongs to one of the non-requirements defined in the OPM model, an OPM-based provenance system can be useful for a scientific workflow whose workflow tasks are subworkflows enacted by different scientific workflow management systems. An example of such a workflow is *GENOMEFLOW* [80]. In this scenario, provenance from different scientific workflow management systems needs to be integrated, and our OPMPROV system can be used for this purpose. OPMPROV is fully compliant with the OPM model and can store provenance generated by different scientific workflow management systems that are able to record OPM-compliant provenance.

In particular, in the Third Provenance Challenge [15], different scientific workflow management systems, including Kepler, Taverna, and Swift, have shown the capability to export OPM-compliant provenance data by means of a mapping between the proprietary models and the OPM model; such heterogeneous provenance from different workflow management systems can be integrated in OPMPROV. In our work, we are particularly interested in using relational database technologies to store, reason, and query OPM-compliant provenance data. Since relational databases are not specifically designed for inferences, we aim to investigate if we can use recursive views and SQL queries alone to perform provenance reasoning.

4.2 Database Schema

We design a relational database schema for our OPMPROV store to suport both prospective and retrospective provenance. Based on our provenance model depicted in Figure 3.1, we translate the E-R diagram into the relational database schema for prospective and retrospective provenance.

4.2.1 Prospective Provenance Database Schema

As shown in Figure 4.1, we define nine relations to store prospective provenance: *Workflow*, *Task*, *Performer*, *Port*, *Contains*, *Performs*, *IsConnectedTo*, *PerformerPartOfWorkflow*, and *TaskPartOfWorkflow*. The primary keys of the relations are underlined. For example, *(ParentTaskId, ChildTaskId)* is the composite primary key of the *Contains* relation. The attributes of relations that

www.manaraa.com

// relational schema for prospective provenance

1. Workflow (WorkflowId, Description)
2. Task (TaskId, TaskName, TaskType)
3. Performer (PFId, PFName)
4. Port (PortId, TaskId, PortName, PortType)
5. Contains (ParentTaskId, ChildTaskId)
6. Performs (PFId, TaskId)
7. IsConnectedTo (SourcePortId, DestinationPortId)
8. TaskPartOfWorkflow(TaskId, WorkflowId)
9. PerformerPartOfWorkflow(PFId, WorkflowId)

// relational schema for retrospective provenance

1. OPMGraph (OPMGraphId, WorkflowId)
2. OPMGraphAnnotation (OPMGraphId, Property, Value)
3. WasInputTo (OPMGraphId, ArtifactId, PortId)
4. WasOutputBy (OPMGraphId, ArtifactId, PortId)
5. Artifact (OPMGraphId, ArtifactId, Value)
6. Process (OPMGraphId, ProcessId, Value, TaskId)
7. Agent (OPMGraphId, AgentId, Value, PFId)
8. Used (OPMGraphId, ProcessId, Role, ArtifactId, OTimeLower, OTimeUpper)
9. WasGeneratedBy (OPMGraphId, ArtifactId, Role, ProcessId, OTimeLower, OTimeUpper)
10. WasControlledBy (OPMGraphId, ProcessId, Role, AgentId, OTimeStartLower, OTimeStartUpper, OTimeEndLower, OTimeEndUpper)
11. WasDerivedFrom (OPMGraphId, EffectArtifactId, CauseArtifactId, OTimeLower, OTimeUpper)
12. ExplicitWasTriggeredBy (OPMGraphId, EffectProcessId, CauseProcessId, OTimeLower, OTimeUpper)
13. ArtifactHasAccount (OPMGraphId, ArtifactId, Account)
14. ProcessHasAccount (OPMGraphId, ProcessId, Account)
15. AgentHasAccount (OPMGraphId, AgentId, Account)
16. UsedHasAccount (OPMGraphId, ProcessId, Role, ArtifactId, Account)
17. WasGeneratedByHasAccount (OPMGraphId, ArtifactId, Role, ProcessId, Account)
18. WasControlledByHasAccount (OPMGraphId, ProcessId, Role, AgentId, Account)
19. WasDerivedFromHasAccount (OPMGraphId, EffectArtifactId, CauseArtifactId, Account)
20. ExplicitWasTriggeredByHasAccount (OPMGraphId, EffectProcessId, CauseProcessId, Account)
21. ArtifactAnnotation (OPMGraphId, ArtifactId, Property, Value)
22. ProcessAnnotation (OPMGraphId, ProcessId, Property, Value)
23. AgentAnnotation (OPMGraphId, AgentId, Property, Value)
24. UsedAnnotation (OPMGraphId, ProcessId, Role, ArtifactId, Property, Value)
25. WasGeneratedByAnnotation (OPMGraphId, ArtifactId, Role, ProcessId, Property, Value)
26. WasControlledByAnnotation (OPMGraphId, ProcessId, Role, AgentId, Property, Value)
27. WasDerivedFromAnnotation (OPMGraphId, EffectArtifactId, CauseArtifactId, Property, Value)
28. ExplicitWasTriggeredByAnnotation (OPMGraphId, EffectProcessId, CauseProcessId, Property, Value)
29. WasTriggeredBy (OPMGraphId, EffectProcessId, CauseProcessId, Account, OTimeLower, OTimeUpper) // view
30. MultiStepWasDerivedFrom (OPMGraphId, EffectArtifactId, CauseArtifactId, Account) // view
31. MultiStepWasTriggeredBy (OPMGraphId, EffectProcessId, CauseProcessId, Account) // view
32. MultiStepUsed (OPMGraphId, ProcessId, ArtifactId, Account) // view
33. MultiStepWasGeneratedBy (OPMGraphId, ArtifactId, ProcessId, Account) // view

Figure 4.1: The database schema for the OPMPROV store.

correspond to relationship types in the E-R diagram represent roles of the relationship with participating entity types, and therefore these are also foreign keys. For example, *ParentTaskId* and *ChildTaskId* in relation *Contains* are two foreign keys that reference *TaskId* in relation *Task*. Since, in the E-R diagram, entity type *Port* participates in relationship type *PartOf* exactly once, instead

www.manaraa.com

of creating a separate relation for this relationship type, we add attribute *TaskId* into relation *Port*. *TaskId* in relation *Port* is a foreign key that references *TaskId* in relation *Task*.

4.2.2 Retrospective Provenance Database Schema

Figure 4.1 also defines 33 relations for retrospective provenance, where the first 28 of them are materialized relations and the remaining five are non-materialized views. Relations *OPMGraph*, *Artifact*, *Process*, *Agent*, *Used*, *WasGeneratedBy*, *WasControlledBy*, *WasDerivedFrom*, *ExplicitWasTriggeredBy*[*], *WasOutputBy*, and *WasInputTo* are directly derived from the E-R diagram. To handle the set-valued attributes *Account* and *Annotation*, additional relations are introduced, such as the corresponding relations *xxxHasAccount* and *xxxAnnotation*. We restrict that each row in relations *Artifact*, *Process*, *Agent*, *Used*, etc. has at least one account and therefore at least one row in the corresponding *xxxHasAccount* relations. This participation constraint eliminates the burden of dealing with missing values when computing relational joins and can be efficiently ensured on the data insertion stage by introducing a default account. The primary keys of these 28 relations are underlined in the figure. Relations *OPMGraph*, *Process*, and *Agent* have foreign keys *WorkflowId*, *TaskId*, and *PFId* that reference relations *Workflow*, *Task*, and *Performer*, respectively, to maintain the *InstanceOf* relationship. For example, relation *Process* has *(OPMGraphId, ProcessId)* as the primary key and *(OPMGraphId, TaskId)* as the foreign key referencing relations *OPMGraph* and *Task*, respectively. The *ProcessHasAccount* relation has *(OPMGraphId, ProcessId, Account)* as the primary key and *(OPMGraphId, ProcessId)* as the foreign key referencing relations *OPMGraph* and *Process*, respectively. Similarly, the *Used* relation has *(OPMGraphId, ProcessId, ArtifactId, Role)* as the composite primary key and the *UsedHasAccount* relation has the primary key *(OPMGraphId, ProcessId, ArtifactId, Role, Account)* and foreign key *(OPMGraphId, ProcessId, ArtifactId, Role)* referencing relation *Used*.

[*]Note that this relation corresponds to relationship type *WasTriggeredBy* in the E-R diagram, but we name it *ExplicitWasTriggeredBy* to differentiate from non-materialized view *WasTriggeredBy* which can be inferred from relations *Used*, *WasGeneratedBy*, and *ExplicitWasTriggeredBy*.

www.manaraa.com

Non-materialized views in our database schema are shown as the last five relations in Figure 4.1. View *WasTriggeredBy* implements the one-step inference rule defined in the OPM model [18], and views *MultiStepWasDerivedFrom*, *MultiStepWasTriggeredBy*, *MultiStepUsed*, and *MultiStepWasGeneratedBy* implement the multi-step edges presented in the OPM model [18]. The semantics and implementation of these recursive views are further discussed in Section 4.4.

4.3 Data Mapping Algorithm

To insert provenance data into OPMPROV, we design an efficient data mapping algorithm that shreds XML documents, which conform to the XML schema specification for the OPM model [13], into relational tuples and stores them into the database. The algorithm is presented in Algorithm 5. First, *OPMXMLInsert* parses an input XML document and constructs its Document Object Model (DOM) tree. Next, the algorithm iterates over all nodes in the tree with the *process* name, extracting process identifiers from their attribute nodes, values and accounts from the child nodes of the process nodes. At the end of each iteration, once the information about a single process is collected, the algorithm inserts a tuple with the OPM graph identifier, process identifier, and process value into database relation *Process*. Furthermore, if the process does not belong to any account, a default account is assigned to the process. For each account, a tuple with OPM graph identifier, process identifier, and account is inserted into relation *ProcessHasAccount*. The algorithm defines similar loops that iterate over all the *artifact* and *agent* nodes and insert them into the database. In the figure, we also show our pseudocode for the insertion of the *used* nodes. For each such node, its child nodes are visited to extract the information about a process identifier from attribute *id* of the *effect* node, role from attribute *value* of the *role* node, artifact identifier from attribute *id* of the *cause* node, accounts from attributes *id* of the *account* nodes, and time annotations from the *noLaterThan* and *noEarlierThan* child nodes of the *time* node. If nodes with names *role* and *account* are not found, default values are assigned, which are customizable. The algorithm inserts a tuple with the OPM graph identifier, process identifier, role, artifact identifier, and time annotations into relation *Used*. For each account, a tuple with the OPM graph identifier, process identifier,

www.manaraa.com

Algorithm 5 *OPMXMLInsert*

```
1:  input: OPM-compliant XML document X (conforms to the
    XML schema http://openprovenancemodel.org/
    model/v1.01.1), OPM graph identifier OPMGraphId
2:  output: X is inserted into the relational database
3:  begin
4:  parse X into DOM tree T;
    // inserting "process" elements
5:  for each e in T.getElementsByTagName("process") do
6:      processId = e.attributes.getNamedItem("id").value;
7:      value = null;
8:      accounts = empth list;
9:      for each c in e.childNodes do
10:         if c.nodeName == "value" then
11:             value = c.nodeValue;
12:         else  // c.nodeName == "account"
13:             accounts.add(c.nodeValue);
14:         end if
15:     end for
16:     insert tuple (OPMGraphId, processId, value) into table
        Process;
17:     if account is empty then
18:         accounts.add("default");
19:     end if
20:     for each account in accounts do
21:         insert tuple (OPMGraphId, processId, account) into
            table ProcessHasAccount;
22:     end for
23: end for
24: //inserting "artifact" elements
25: for each e in T.getElementsByTagName("artifact") do
    // ...
    // instructions are similar to the process "process" element in-
    sertion
26: end for
27: //inserting "agent" elements
28: for each e in T.getElementsByTagName("agent") do
    // ...
    // instructions are similar to the process "process" element in-
    sertion
29: end for
    //inserting "used" elements
30: for each e in T.getElementsByTagName("used") do
31:     for each c in e.childNodes do
32:         processId = null; role = "default"; artifactId = null;
33:         oTimeLower = null; oTimeUpper = null;
34:         accounts = empty list;
35:         switch c.nodeName
36:         case "effect":
37:           processId = c.attributes.getNamedItem("id").value;
38:         break
39:         case "role":
40:           role = c.attributes.getNamedItem("value").value;
41:         break
42:         case "cause":
43:           artifactId = c.attributes.getNamedItem("id").value;
44:         break
45:         case "account":
46:          accounts.add(c.attributes.getNamedItem("id").value;
47:         break
48:         case "time":
49:         for each t in c.childNodes do
50:             if t.nodeName =="noLaterThan" then
51:                 oTimeLower = t.nodeValue;
52:             end if
53:             if t.nodeName =="noEarlierThan" then
54:                 oTimeUpper = t.nodeValue;
55:             end if
56:         end for
57:         break
58:         end switch
59:     end for
60:     insert tuple (OPMGraphId, processId, role, artifactId,
        oTimeLower, oTimeUpper) into table Used;
61:     if accounts is empty then
62:         accounts.add("default");
63:     end if
64:     for each account in accounts do
65:         insert tuple (OPMGraphId, processId, role, artifactId,
            account) into table UsedHasAccount;
66:     end for
67: end for
    // ...
    // insert other causal dependency elements wasGeneratedBy,
    wasDerivedFrom, wasTriggeredBy, and wasControlledBy in a
    similar fashion
68: end
```

role, artifact identifier, and account is inserted into relation *UsedHasAccount*. Other nodes that define the causal dependencies of the OPM model, including *wasGeneratedBy*, *wasDerivedFrom*, *wasTriggeredBy*, and *wasControlledBy*, are processed using a similar strategy.

4.4 Provenance Reasoning and Querying

In this section, we report on how the inference rules defined in the OPM model [18] can be expressed in SQL (IBM DB2 dialect) and implemented in an RDBMS directly, eliminating the need for an external inference engine. We then showcase the querying capabilities of OPMPROV to

www.manaraa.com

answer sample queries defined in the Third Provenance Challenge [15]. IBM DB2 is selected for our presentation for its support for the definition of recursive views, which can be referred in SQL queries. A similar recursive behavior can be achieved using the WITH clause and common table expressions defined in SQL-99. However, repeating the WITH clause in every SQL query can result in more verbose presentation. Even though we present OPMPROV in the context of DB2, with minor syntactic changes in SQL definitions, our approach can be used with other major RDBMSs, including Oracle, SQLServer, and PostgreSQL (MySQL provides no support for common table expressions).

```
CREATE VIEW WasTriggeredBy AS (
 (SELECT U.OPMGraphId, U.ProcessId as EffectProcessId, G.ProcessId as CauseProcessId,
         UA.Account as Account, U.OTimeLower, U.OTimeUpper
  FROM WasGeneratedBy G, Used U, UsedHasAccount UA
  WHERE U.OPMGraphId = G.OPMGraphId AND U.OPMGraphId = UA.OPMGraphId AND
        U.ArtifactId = G.ArtifactId AND U.ProcessId = UA.ProcessId AND
        U.ArtifactId = UA.ArtifactId AND U.Role = UA.Role)
 UNION
 (SELECT U.OPMGraphId, U.ProcessId as EffectProcessId, G.ProcessId as CauseProcessId,
         GA.Account as Account, U.OTimeLower, U.OTimeUpper
  FROM WasGeneratedBy G, Used U, WasGeneratedByHasAccount GA
  WHERE U.OPMGraphId = G.OPMGraphId AND G.OPMGraphId = GA.OPMGraphId AND
        U.ArtifactId = G.ArtifactId AND G.ArtifactId = GA.ArtifactId AND
        G.ProcessId = GA.ProcessId AND G.Role = GA.Role)
 UNION
 (SELECT T.OPMGraphId, T.EffectProcessId, T.CauseProcessId, TA.Account, T.OTimeLower, T.OTimeUpper
  FROM ExplicitWasTriggeredBy T, ExplicitWasTriggeredByHasAccount TA
  WHERE T.OPMGraphId = TA.OPMGraphId AND T.EffectProcessId = TA.EffectProcessId
        AND T.CauseProcessId = TA.CauseProcessId) );
```

Figure 4.2: An SQL view: one-step inference *WasTriggeredBy*.

4.4.1 Reasoning for One-Step Inferences

The OPM model defines completion rules (i.e., one-step inferences) for causal dependencies *WasTriggeredBy* and *WasDerivedFrom*. In particular, *WasTriggeredBy* edge in the OPM model can be inferred from the existence of the *Used* and *WasGeneratedBy* edges. If one process generated an artifact that was used by another process, then the latter was triggered by the former. The SQL view that implements this logic in a relational database are shown in Figure 4.2. The *WasTriggeredBy* view derives which process (attribute *EffectProcessId*) was triggered by another process (attribute

www.manaraa.com

CauseProcessId) using *Used* and *WasGeneratedBy* facts with the same artifact. The account information in this view is derived from both *UsedHasAccount* and *WasGeneratedByHasAccount* relations. In addition, the view need to accommodate explicit *wasTriggeredBy* edges stored in the *ExplicitWasTriggeredBy* relation. As a result, the view definition (1) joins relations *WasGeneratedBy*, *Used*, and *UsedHasAccount*, (2) joins relations *WasGeneratedBy*, *Used*, and *WasGeneratedByHasAccount*, (3) unions the results of the joins, and (4) unions inferred *wasTriggeredBy* edges with explicit ones. OPM's time annotations for *WasTriggeredBy* are captured by the *OTimeLower* and *OTimeUpper* attributes and derived from the *Used* relation. *OTimeLower* and *OTimeUpper* define a time interval when an artifact was used by a process and therefore when this process was triggered by another process.

4.4.2 Reasoning for Multi-Step Inferences

The OPM model defines multi-step inferences for four multi-step edges *WasDerivedFrom**, *WasTriggeredBy**, *Used**, and *WasGeneratedBy**. OPMPROV supports these multi-step inferences via recursive views and SQL queries. As shown in Figure 4.3, the *MultiStepWasDerivedFrom* view is defined as the union of two auxiliary recursive views that only differ in how they project account information. Each auxiliary view has a non-recursive subquery that retrieves tuples from the *WasDerivedFromHasAccount* relation and a recursive part that joins the *WasDerivedFromHasAccount* relation with the recursive view itself, such that if the first artifact was derived from the second artifact and the second artifact was derived from the third one, the view infers that the first artifact was also derived from the third artifact. In Figure 4.4, the *MultiStepWasGeneratedBy* view is defined as the union of three subqueries returning: (1) given facts about which artifacts was generated by which processes from relation *WasGeneratedByHasAccount*, (2) entailments about artifacts and processes inferred via the join of relation *WasGeneratedByHasAccount* and recursive view *MultiStepWasDerivedFrom* with the projection of account information from the former, and (3) entailments about artifacts and processes inferred via the join of relation *WasGeneratedByHasAccount* and recursive view *MultiStepWasDerivedFrom* with the projection of account

www.manaraa.com

information from the latter. Similarly, view *MultiStepUsed* (see Figure 4.5) is defined using relation *UsedHasAccount* and recursive view *MultiStepWasDerivedFrom*, and view *MultiStepWasTriggeredBy* (see Figure 4.6) is defined using relation *UsedHasAccount*, *WasGeneratedByHasAccount*, and *MultiStepWasDerivedFrom*.

```
CREATE VIEW MultiStepWasDerivedFrom1 (OPMGraphId, EffectArtifactId, CauseArtifactId, Account) AS (
  SELECT DA1.OPMGraphId, DA1.EffectArtifactId, DA1.CauseArtifactId, DA1.Account
  FROM WasDerivedFromHasAccount DA1
  UNION ALL
  SELECT DA2.OPMGraphId, DA2.EffectArtifactId, TD.CauseArtifactId, DA2.Account
  FROM WasDerivedFromHasAccount DA2, MultiStepWasDerivedFrom1 TD
  WHERE DA2.OPMGraphId = TD.OPMGraphId AND DA2.CauseArtifactId = TD.EffectArtifactId );

CREATE VIEW MultiStepWasDerivedFrom2 (OPMGraphId, EffectArtifactId, CauseArtifactId, Account) AS (
  SELECT DA1.OPMGraphId, DA1.EffectArtifactId, DA1.CauseArtifactId, DA1.Account
  FROM WasDerivedFromHasAccount DA1
  UNION ALL
  SELECT DA2.OPMGraphId, DA2.EffectArtifactId, TD.CauseArtifactId, TD.Account
  FROM WasDerivedFromHasAccount DA2, MultiStepWasDerivedFrom2 TD
  WHERE DA2.OPMGraphId = TD.OPMGraphId AND DA2.CauseArtifactId = TD.EffectArtifactId );

CREATE VIEW MultiStepWasDerivedFrom AS (
  SELECT OPMGraphId, EffectArtifactId, CauseArtifactId, Account
  FROM MultiStepWasDerivedFrom1
  UNION
  SELECT OPMGraphId, EffectArtifactId, CauseArtifactId, Account
  FROM MultiStepWasDerivedFrom2 );
```

Figure 4.3: An SQL view: multi-step inference *WasDerivedFrom**.

```
CREATE VIEW MultiStepWasGeneratedBy (OPMGraphId, ArtifactId, ProcessId, Account) AS (
  SELECT GA1.OPMGraphId, GA1.ArtifactId, GA1.ProcessId, GA1.Account
  FROM WasGeneratedByHasAccount GA1
  UNION
  SELECT GA2.OPMGraphId, TD.EffectArtifactId as ArtifactId, GA2.ProcessId, GA2.Account
  FROM WasGeneratedByHasAccount GA2, MultiStepWasDerivedFrom TD
  WHERE GA2.OPMGraphId = TD.OPMGraphId AND GA2.ArtifactId = TD.CauseArtifactId
  UNION
  SELECT GA2.OPMGraphId, TD.EffectArtifactId as ArtifactId, GA2.ProcessId, TD.Account
  FROM WasGeneratedByHasAccount GA2, MultiStepWasDerivedFrom TD
  WHERE GA2.OPMGraphId = TD.OPMGraphId AND GA2.ArtifactId = TD.CauseArtifactId );
```

Figure 4.4: An SQL view: multi-step inference *WasGeneratedBy**.

4.4.3 SQL-Based Provenance Querying

The Third Provenance Challenge [15] defined 16 provenance queries for the Load Workflow from the Pan-STARRS project [16], including three core queries (CQ) and 13 optional queries (OQ). To

www.manaraa.com

```
CREATE VIEW MultiStepUsed (OPMGraphId, ProcessId, ArtifactId, Account) AS (
  SELECT UA1.OPMGraphId, UA1.ProcessId, UA1.ArtifactId, UA1.Account
  FROM UsedHasAccount UA1
  UNION
  SELECT TD.OPMGraphId, UA2.ProcessId, TD.CauseArtifactId as ArtifactId, UA2.Account
  FROM MultiStepWasDerivedFrom TD, UsedHasAccount UA2
  WHERE TD.OPMGraphId = UA2.OPMGraphId AND TD.EffectArtifactId = UA2.ArtifactId
  UNION
  SELECT TD.OPMGraphId, UA2.ProcessId, TD.CauseArtifactId as ArtifactId, TD.Account
  FROM MultiStepWasDerivedFrom TD, UsedHasAccount UA2
  WHERE TD.OPMGraphId = UA2.OPMGraphId AND TD.EffectArtifactId = UA2.ArtifactId );
```

Figure 4.5: An SQL view: multi-step inference *Used**.

```
CREATE VIEW MultiStepWasTriggeredBy (OPMGraphId, EffectProcessId, CauseProcessId, Account) AS (
  SELECT T1.OPMGraphId, T1.EffectProcessId, T1.CauseProcessId, T1.Account
  FROM WasTriggeredBy T1
  UNION
  SELECT UA1.OPMGraphId, UA1.ProcessId, GA1.ProcessId, UA1.Account
  FROM UsedHasAccount UA1, WasGeneratedByHasAccount GA1, MultiStepWasDerivedFrom TD
  WHERE UA1.OPMGraphId = TD.OPMGraphId AND UA1.ArtifactId = TD.EffectArtifactId AND
        GA1.OPMGraphId = TD.OPMGraphId AND TD.CauseArtifactId = GA1.ArtifactId
  UNION
  SELECT UA2.OPMGraphId, UA2.ProcessId, GA2.ProcessId, GA2.Account
  FROM UsedHasAccount UA2, WasGeneratedByHasAccount GA2, MultiStepWasDerivedFrom TD
  WHERE UA2.OPMGraphId = TD.OPMGraphId AND UA2.ArtifactId = TD.EffectArtifactId AND
        GA2.OPMGraphId = TD.OPMGraphId AND TD.CauseArtifactId = GA2.ArtifactId );
```

Figure 4.6: An SQL view: multi-step inference *WasTriggeredBy**.

demonstrate querying capabilities of OPMPROV, we express some of these queries in SQL that is executable over our database schema. In particular, in Figure 4.7, we present 15 provenance queries in English and SQL. In the following, we describe some of these more advanced queries that use reasoning via the recursive views. First, query CQ1, which asks for CSV files that contributed to a given detection, uses view *MultiStepWasGeneratedBy* to find all process identifiers that contributed to the generation of the artifact with the value "detectID". It then retrieves artifacts that are CSV files used by those processes. The *MultiStepWasGeneratedBy* view is also used in a similar fashion in query OQ8. Second, query OQ4, which asks why a certain entry is presented in a database, uses view *MultiStepWasTriggeredBy* to find all processes that directly or indirectly triggered a process that generated an artifact with a given value (e.g., "ccdID"). It then returns those process values along with their count. This view is also used in queries CQ3 and OQ13. Finally, query OQ13, which asks for artifact and process dependency views, can be directly satisfied by two SQL queries

www.manaraa.com

that retrieve all data from views *MultiStepWasTriggeredBy* and *MultiStepWasDerivedFrom*. None of the presented queries required to access view *MultiStepUsed*.

CQ1:	For a given detection, which CSV files contributed to it?
SQL:	SELECT DISTINCT A2.Value FROM Artifact A2, Used U, (SELECT DISTINCT TG.OPMGraphId, TG.ProcessId FROM MultiStepWasGeneratedBy TG, Artifact A WHERE TG.OPMGraphId = A.OPMGraphId AND TG.ArtifactId = A.ArtifactId AND A.Value = '261887437010025730') As Pv WHERE U.OPMGraphId = Pv.OPMGraphId AND U.ProcessId = Pv.ProcessId AND U.OPMGraphId = A2.OPMGraphId AND U.ArtifactId = A2.ArtifactId AND A2.Value LIKE '%Detection.csv' AND U.OPMGraphId = '1'
CQ2:	The user considers a table to contain values they do not expect. Was the range check (IsMatchTableColumnRanges) performed for this table?
SQL:	SELECT 'YES' AS Answer FROM (SELECT COUNT(G.ArtifactId) AS Number FROM WasGeneratedBy G WHERE G.ProcessId LIKE '%IsMatchTableColumnRanges%' AND G.OPMGraphId = '1') AS Output WHERE Output.Number > 0
CQ3:	Which operation executions were strictly necessary for the Image table to contain a particular (non-computed) value?
SQL:	SELECT Cp.Value As Operation, SUM(Cp.Number) As Count FROM (SELECT DISTINCT TT.OPMGraphId, TT.CauseProcessId, P.Value, 1 As Number FROM MultiStepWasTriggeredBy TT, Artifact A, Process P, WasGeneratedBy G WHERE TT.OPMGraphId = G.OPMGraphId AND G.OPMGraphId = A.OPMGraphId AND TT.EffectProcessId = G.ProcessId AND G.ArtifactId = A.ArtifactId AND A.Value LIKE '%Image%' AND TT.OPMGraphId = P.OPMGraphId AND TT.CauseProcessId = P.ProcessId) As Cp WHERE Cp.OPMGraphId = '1' GROUP BY Cp.Value
OQ1:	How many tables successfully loaded before the workflow halted due to a failed check?
SQL:	SELECT COUNT(*) AS Count FROM Artifact A, WasGeneratedBy G WHERE A.OPMGraphId = G.OPMGraphId AND A.ArtifactId = G.ArtifactId AND G.ProcessId LIKE 'IsMatchTableColumnRanges%' AND A.Value LIKE '%success%' AND A.OPMGraphId = '1'
OQ3:	How much time expired between a successful IsMatchCSVFileTables test and an unsuccessful IsExistsCSVFile test?
SQL:	SELECT T2.OTimeUpper As StartTime, T1.OTimeUpper As EndTime FROM (SELECT W1.OTimeUpper FROM Artifact A1, WasGeneratedBy W1 WHERE A1.OPMGraphId = W1.OPMGraphId AND A1.ArtifactId = W1.ArtifactId AND W1.ProcessId LIKE '%IsExistsCSVFile%' AND A1.Value LIKE '%halt%' AND W1.OPMGraphId = '1') As T1, (SELECT W2.OTimeUpper FROM Artifact A2, WasGeneratedBy W2 WHERE A2.OPMGraphId = W2.OPMGraphId AND A2.ArtifactId = W2.ArtifactId AND W2.ProcessId LIKE '%IsMatchCSVFileTables%' AND A2.Value LIKE '%success%' AND W2.OPMGraphId = '1') As T2
OQ4:	Why is this entry in the database?
SQL:	SELECT Pv.Value, SUM (Pv.Number) As Count FROM (SELECT DISTINCT TT.OPMGraphId, TT.CauseProcessId, P.Value, 1 As Number FROM MultiStepWasTriggeredBy TT, Artifact A, WasGeneratedBy G, Process P WHERE TT.OPMGraphId = G.OPMGraphId AND TT.EffectProcessId = G.ProcessId AND TT.OPMGraphId = P.OPMGraphId AND TT.CauseProcessId = P.ProcessId AND A.OPMGraphId = G.OPMGraphId AND A.ArtifactId = G.ArtifactId AND A.Value = ' ccdID') AS Pv WHERE Pv.OPMGraphId = '1' GROUP BY Pv.Value
OQ5:	A user executes the workflow many times (say 5 times) over different sets of data (j062941, ..., j062945). He wants to determine, which of the execution halted?
SQL:	SELECT A.OPMGraphId, A.Value AS NameOfDataset FROM Artifact A, WasGeneratedBy W WHERE A.OPMGraphId = W.OPMGraphId AND A.ArtifactId = W.ArtifactId AND A.Value LIKE 'J%halt%'
OQ6:	Determine the step where Halt occurred?
SQL:	SELECT Hp.Value As HaltStep, SUM(Hp.Number) As Count FROM (SELECT DISTINCT P.OPMGraphId, P.ProcessId, P.Value, 1 As Number FROM Artifact A, WasGeneratedBy G, Process P WHERE A.Value LIKE '%halt%' AND A.OPMGraphId = G.OPMGraphId AND A.ArtifactId = G.ArtifactId AND G.OPMGraphId = P.OPMGraphId AND G.ProcessId = P.ProcessId) As Hp WHERE Hp.OPMGraphId = '1' GROUP BY Hp.Value
OQ7:	Determine data and associated granularities of the data being processed, when halt occurred?
SQL:	SELECT DISTINCT A2.ArtifactId, A2.Value FROM Artifact A1, WasGeneratedBy G, Artifact A2, Used U WHERE A1.Value LIKE '%halt%occurred%' AND A1.OPMGraphId = G.OPMGraphId AND A1.ArtifactId = G.ArtifactId AND U.OPMGraphId = G.OPMGraphId AND U.ProcessId = G.ProcessId AND A2.OPMGraphId = U.OPMGraphId AND A2.ArtifactId = U.ArtifactId AND A2.OPMGraphId = '1'
OQ8:	Which steps were completed successfully before the halt occurred?
SQL:	SELECT Sp.Value As Step, SUM (Sp.Number) As Count FROM (SELECT DISTINCT TW.OPMGraphId, TW.ProcessId, P.Value, 1 As Number FROM MultiStepWasGeneratedBy TW, Artifact A, Process P WHERE TW.OPMGraphId = A.OPMGraphId AND TW.ArtifactId = A.ArtifactId AND A.Value LIKE '%success%' AND TW.OPMGraphId = P.OPMGraphId AND TW.ProcessId = P.ProcessId) As Sp WHERE Sp.OPMGraphId = '1' GROUP BY Sp.Value
OQ9:	Which steps were not executed because of halt?
SQL:	SELECT DISTINCT OT.TaskId, OT.TaskName FROM Task OT WHERE NOT EXISTS (SELECT T.TaskId FROM (SELECT DISTINCT TW.ProcessId, TaskId FROM TransitiveClosureWasGeneratedBy TW, Artifact A, Process P, WasGeneratedBy G WHERE TW.OPMGraphId = A.OPMGraphId AND TW.ArtifactId = A.ArtifactId AND A.Value LIKE '%success%' AND TW.OPMGraphId = P.OPMGraphId AND TW.ProcessId = P.ProcessId) AS SC, Task T WHERE T.TaskId = SC.TaskID AND T.TaskId = OT.TaskId)
OQ10:	For a workflow execution, determine the user inputs?
SQL:	SELECT A.Value FROM Used U, Artifact A WHERE U.OPMGraphId = A.OPMGraphId AND U.ArtifactId = A.ArtifactId AND NOT EXISTS (SELECT * FROM WasGeneratedBy G WHERE G.OPMGraphId = A.OPMGraphId AND G.ArtifactId = A.ArtifactId) AND A.OPMGraphId = '1'
OQ11:	For a workflow execution, determine steps that required user inputs?
SQL:	SELECT DISTINCT P.ProcessId, P.Value FROM Used U, Artifact A, Process P WHERE U.OPMGraphId = A.OPMGraphId AND U.ArtifactId = A.ArtifactId AND U.OPMGraphId = P.OPMGraphId AND U.ProcessId = P.ProcessId AND NOT EXISTS (SELECT * FROM WasGeneratedBy G WHERE G.OPMGraphId = A.OPMGraphId AND G.ArtifactId = A.ArtifactId) AND P.OPMGraphId = '1'
OQ12:	For a workflow execution that halted, which files where processed successfully?
SQL:	SELECT DISTINCT A2.ArtifactId, A2.Value FROM WasGeneratedBy G, Artifact A1, Used U, Artifact A2 WHERE G.OPMGraphId = A1.OPMGraphId AND G.ArtifactId = A1.ArtifactId AND A1.Value LIKE '%success%' AND U.OPMGraphId = G.OPMGraphId AND U.ProcessId = G.ProcessId AND A2.OPMGraphId = U.OPMGraphId AND A2.ArtifactId = U.ArtifactId AND A2.Value LIKE '%CSVFileEntry%' AND A2.OPMGraphId = '1'
OQ13:	Display the following provenance views: data dependency view and step dependency view.
SQL:	SELECT EffectProcessId, CauseProcessId FROM MultiStepWasTriggeredBy WHERE OPMGraphId = '1'; SELECT EffectArtifactId, CauseArtifactId FROM MultiStepWasDerivedFrom WHERE OPMGraphId = '1';

Figure 4.7: Provenance queries for the Third Provenance Challenge questions.

www.manaraa.com

4.5 Experimental Study

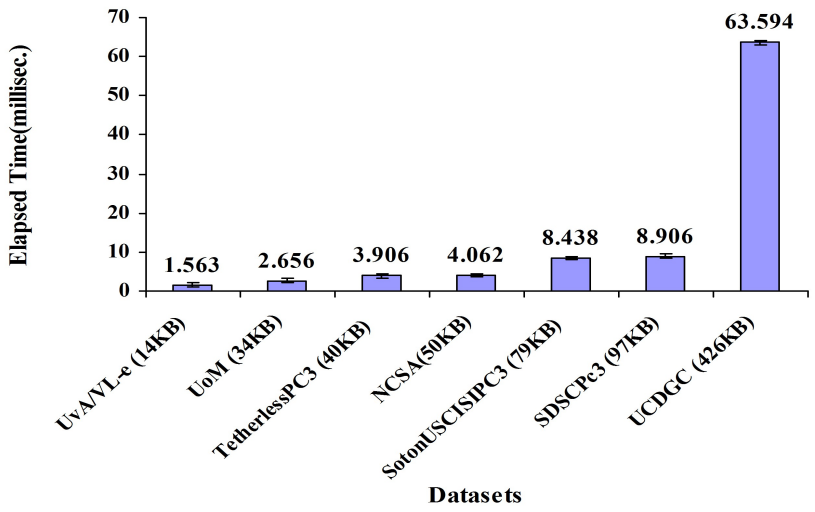
In this section, we report on our experimental study that explored the performance of the data insertion and provenance challenge queries over various datasets. The proposed database schema for OPMPROV was created in RDBMS DB2 (v9.7.0.441), and the *OPMXMLInsert* algorithm was implemented using the C# programming language for data insertion performance experiments. The experiments presented below were conducted on a PC with one 2.4 GHz Pentium IV processor and 1 GB main memory, running the Windows XP Professional operating system. In all the experiments, we show the results as the average of 100 trials.

4.5.1 Data Insertion Performance Experiments

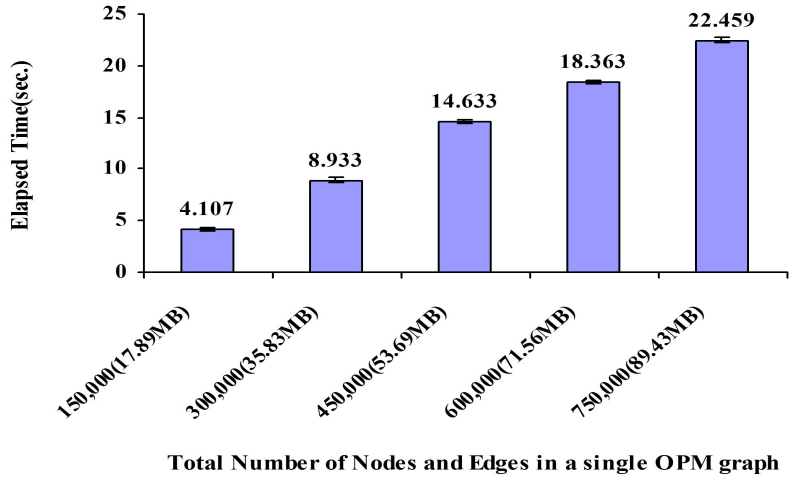
To perform data insertion, we created the tables and views described in Figure 4.1 and selected seven OPM-compliant XML documents generated by different participants of the Third Provenance Challenge [15], including the datasets posted by teams UvA/VL-e (University of Amsterdam), UoM (University of Manchester), TetherlessPC3 (Rensselaer Polytechnic Institute/Tetherless World Constellation), NCSA (National Center for Supercomputing Applications), SotonUSCISIPc3 (University of Southampton and USC/ISI), SDSCPc3 (San Diego Supercomputer Center), and UCDGC (UC Davis Genome Center). The data insertion performance for these datasets is reported in Figure 4.8(a), where the datasets are shown in the ascending order of their sizes. The results for data insertion showed to load all the datasets in less than 100 milliseconds.

To explore the data insertion performance and scalability on larger datasets, we generated five XML documents according to the OPM XML schema [13] , which represent the OPM graphs with varying complexity in which the total number of nodes and edges is 150,000 (17.89 MB), 300,000 (35.83 MB), 450,000 (53.69 MB), 600,000 (71.56 MB), and 750,000 (89.43 MB), respectively. The results for these datasets are reported in Figure 4.8(b).The data loading performance revealed the stable linear scalability with the data insertion rate of around four megabytes per second.

www.manaraa.com



(a) Data insertion performance accross different datasets of the Third Provenance Challenge.



(b) Data insertion performance on the OPM graphs with varying complexity.

Figure 4.8: Data insertion performance over various datasets.

4.5.2 Provenance Query Performance Experiments

OPMPROV was evaluated on 15 queries defined by the Third Provenance Challenge and expressed in SQL over our database schema as shown in Figure 4.7. Note that no participants have provided an answer to OQ2 in the Third Provenance Challenge [15]; therefore, our experiments exclude OQ2. The queries were executed over two different datasets stored into the OPMPROV system.

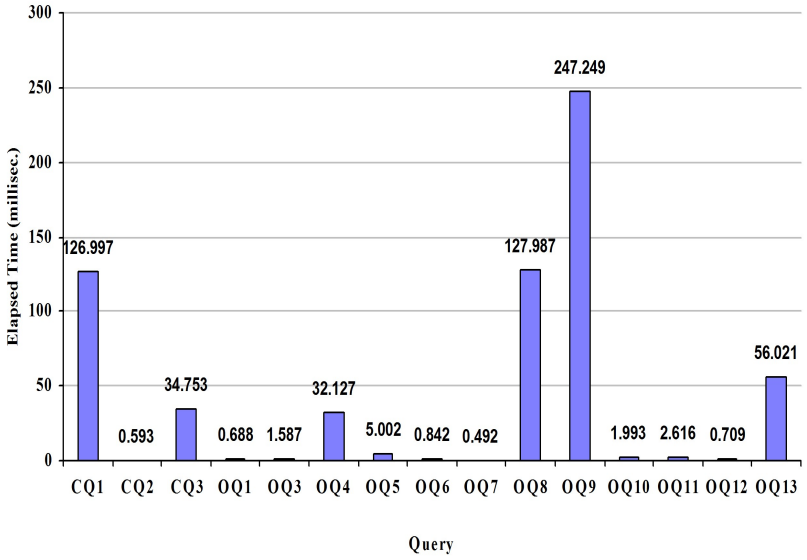
www.manaraa.com

Since the two datasets only contained retrospective provenance compliant with OPM, we additionally generated related prospective provenance that was also stored into the provenance store. The query performance experiments on the UCDGC dataset (426 KB) and the TetherlessPC3 dataset (40 KB) are reported in Figures 4.9. The results returned by the queries were compared with those provided by the UCDGC and TetherlessPC3 teams to ensure correctness [15]. Overall, the query evaluation in OPMPROV showed to be very efficient, returning results within a few milliseconds for queries CQ2, OQ1, OQ3, OQ5, OQ6, OQ7, OQ10, OQ11, and OQ12 that did not compute transitive closures and within a few seconds for queries CQ1, CQ3, OQ4, OQ8, OQ9, and OQ13 that required evaluation of recursive views. Non-recursive queries were performed faster on the smaller dataset, however the recursive ones (except OQ13) showed to be faster on the larger dataset. This is explained by the fact that even though the UCDGC XML document was larger in size than the TetherlessPC3 XML document, the number of tuples computed by views *MultiStepWasTriggeredBy* and *MultiStepWasGeneratedBy* was substantially smaller for the UCDGC dataset. On the other hand, the TetherlessPC3 dataset contained no XML elements with the *wasDerivedFrom* name, resulting in empty relation *WasDerivedFrom* and empty view *MultiStepWasDerivedFrom*. Therefore, since OQ13 required the evaluation of the two SQL queries that accessed views *MultiStepWasTriggeredBy* and *MultiStepWasDerivedFrom*, OQ13 was just a few milliseconds faster on the smaller dataset.

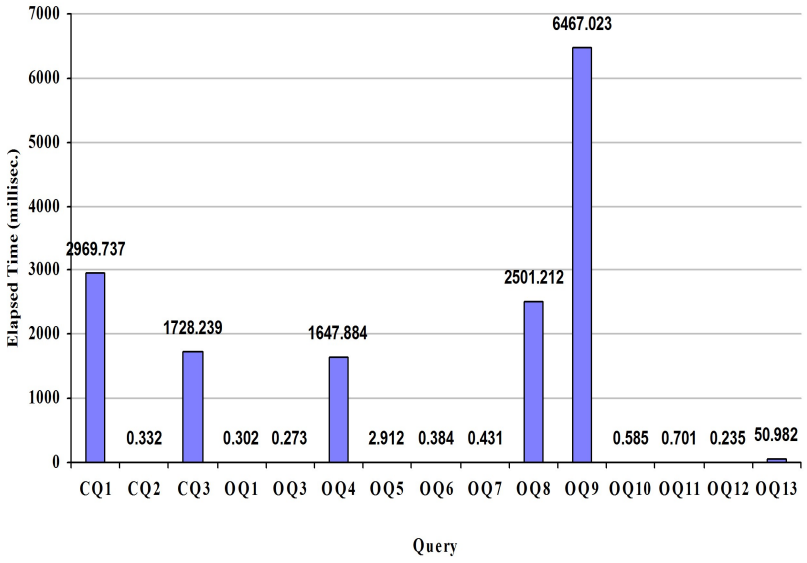
Moreover, to explore the scalability of queries CQ1, CQ3, OQ4, and OQ8 that required computation of recursive views, OPMPROV was evaluated on larger datasets. The response times for these queries and four database size settings are reported in Figure 4.10. Overall, the queries with recursive views showed satisfactory performance, returning results within around 40 seconds for the provenance dataset with 250,000 nodes and edges (66.25 MB), however additional experiments involving larger datasets and server-class machines are required in the future.

Finally, we conducted an experiment to compare the query performance of OPMPROV with the query performance of Karma [61]. While OPMPROV's storage facility was solely based on relational database technology, Karma stored an XML document as a value of an XML-typed

www.manaraa.com



(a) The query performance on the UCDGC dataset.



(b) The query performance on the TetherlessPC3 dataset.

Figure 4.9: OPMPROV query performance over two different datasets.

column in a relational table and, as a result, used both SQL and XPath queries to answer provenance queries. Another significant difference between the two systems was their inference support. While OPMPROV used recursive views to implement OPM's multi-step inference rules, Karma had no inference support for multi-step edges defined in the OPM model [18]. Therefore, due to these

www.manaraa.com

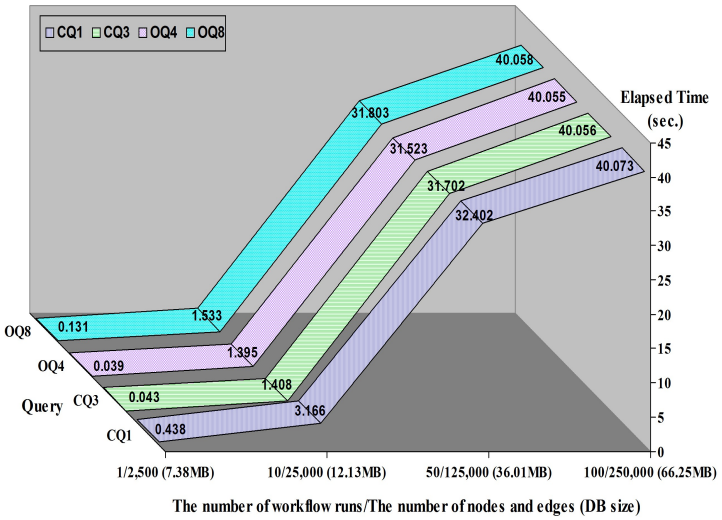


Figure 4.10: OPMPROV query performance for recursive views.

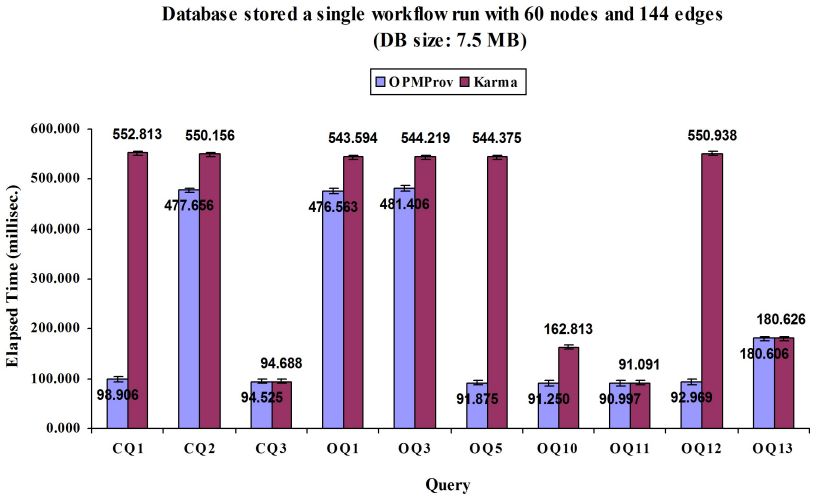


Figure 4.11: Query performance over OPMPROV and Karma.

implementation differences, we report our comparison observation without further analysis. In this experiment, we evaluated queries CQ1, CQ2, CQ3, OQ1, OQ3, OQ5, OQ10, OQ11, OQ12, and OQ13 over a single OPM-compliant XML document with 60 nodes and 144 edges that was generated by the Karma team for the Third Provenance Challenge [15]. The experiment results are reported in Figure 4.11. For all queries, the response times of OPMPROV were smaller than those

www.manaraa.com

of Karma, even though Karma did not perform transitive closure inference for queries CQ1 and CQ3.

4.6 Summary

In this chapter, we designed a relational database schema that supports prospective and retrospective provenance. We then proposed an efficient data mapping algorithm, *OPMXMLInsert* that stores OPM-compliant provenance data into our OPMPROV store. Then, we showed that our OPMPROV can sufficiently support provenance reasoning defined in the OPM model using recursive views and SQL queries alone without any additional reasoning engine. To evaluate the performance of OPMPROV, we conducted experiments on data insertion and provenance querying. Our case study demonstrated that OPMPROV could answer all but one query out of the 16 queries defined in the Third Provenance Challenge.

www.manaraa.com

CHAPTER 5
PROVENANCE QUERY LANGUAGE: OPQL

In this chapter, we propose *OPQL*, an OPM-level provenance query language, that is directly defined over the Open Provenance Model (OPM). An *OPQL* query takes an OPM graph as input and produces an OPM graph as output. Therefore, *OPQL* queries are not tightly coupled to the underlying provenance storage strategies.

5.1 The Problem

As discussed previously, most existing systems [61], [25], [28], [54] store provenance data in their provenance stores of proprietary provenance models and conduct provenance querying using query languages, such as SQL, SPARQL, and XQuery over the physical provenance storages (i.e., RDB, RDF, and XML). Such query languages are closely coupled to the underlying provenance storage strategies, and therefore users have to know the structures or schemas of such provenance storages, as well as semantics of provenance models that have been applied to the provenance storages to formulate provenance queries. Moreover, users require the expertise about grammars, syntax, and semantics of such languages to formulate complicated provenance queries. For example, using existing approaches, *provenance lineage queries* (queries for tracking ancestor nodes) often require users to write recursive queries (directly typing recursive statements or using recursive functionality), which are nontrivial.

To address these issues, in this dissertation, we propose *OPQL*, an OPM-level provenance query language that efficiently supports provenance queries. *OPQL* is a graph query language that is directly defined over the OPM model [18], which is a standard provenance model in the community. An *OPQL* query takes one OPM graph as input and produces an OPM graph as output; therefore, *OPQL* queries are not tightly coupled to the underlying storage strategies. In

www.manaraa.com

particular, to design the *OPQL* query language, we define six types of graph patterns, an OPM-based graph algebra based on four *OPQL* operators, and *OPQL* syntax and semantics. To our best knowledge, *OPQL* is a first proposal on the OPM-level provenance query language for scientific workflows. Moreover, to enhance the accessibility and availability of *OPQL*, we provide *OPQL* as a Web service; therefore, users can invoke the *OPQL* Web service to execute *OPQL* queries in a user-friendly GUI, called OPMProVis, where the result of *OPQL* queries is displayed as an OPM graph. To our best knowledge, OPMProv supports the first OPM-compliant provenance querying service for scientific workflows.

5.2 The OPQL Provenance Query Language

In this section, we describe the *OPQL* query language to efficiently support provenance queries. We first formalize the OPM model which is used as a fundamental provenance model for *OPQL*. Next, we define six types of graph patterns which are the main building blocks of an *OPQL* query and an OPM-based graph algebra for *OPQL*. We then propose OPQL syntax and semantics. Finally, we discuss how provenance queries can be expressed in *OPQL*.

5.2.1 Formalizing the OPM Model

The OPM model [18] is a standard provenance model in the community to facilitate and promote provenance interoperability among heterogeneous systems. In essence, the OPM model consists of a directed graph expressing the dependencies (i.e., how "things" depended on others and resulted in specific states). An OPM graph is composed of three types of nodes (i.e., *Artifact*, *Process*, and *Agent*) and five types of edges (i.e., *WasGeneratedBy*, *Used*, *WasDerivedFrom*, *WasTriggeredBy*, and *WasControlledBy*), which represent causal dependencies between nodes. An artifact is an immutable piece of state, a process is an action or a series of actions, and an agent is a contextual entity acting as a catalyst of a process. The five edges are described through the following sample provenance graph.

www.manaraa.com

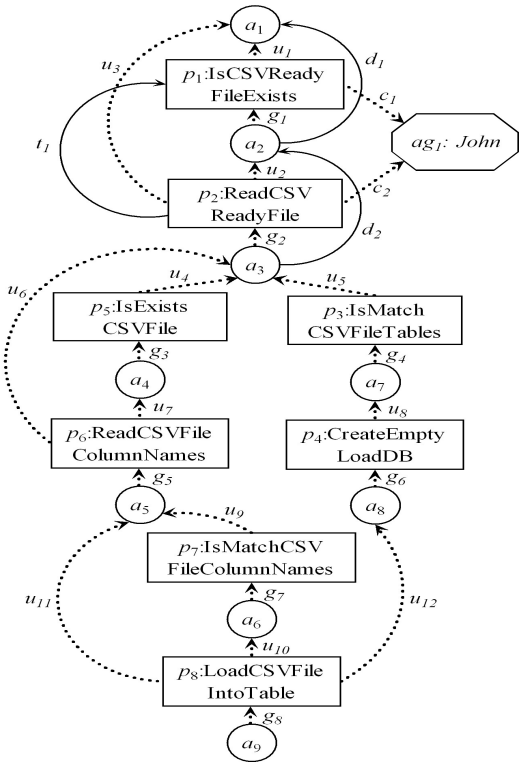


Figure 5.1: A sample OPM graph.

In Figure 5.1 (which is an example OPM graph that can be generated via the execution of the Load Workflow defined in the Third Provenance Challenge [15]), an artifact, process, and agent are represented as an ellipse, rectangle, and octagon shape, respectively, and an edge is represented by an arc and denotes the presence of a causal dependency between the source of the arc (the effect) and the destination of the arc (the cause). As depicted in Figure 5.1, the edges represent the following causal dependencies: (1) edge *Used* (u_1-u_{12}): p_1 *used* a_1, p_2 *used* a_1 and a_2, p_3 *used* a_3, and so on; (2) edge *WasGeneratedBy* (g_1-g_8): a_2 *was generated by* p_1, a_3 *was generated by* p_2, a_4 *was generated by* p_5, and so on; (3) edge *WasDerivedFrom* (d_1, d_2): a_2 *was derived from* a_1 and a_3 *was derived from* a_2; (4) edge *WasTriggeredBy* (t_1): p_2 *was triggered by* p_1; (5) edge *WasControlledBy* (c_1, c_2): p_1 and p_2 *were controlled by* ag_1, respectively. More details on the constituents of the OPM graph can be found in [18].

www.manaraa.com

Based on the OPM model, we formalize an OPM graph as follows. An *OPM graph* OG = (V, E) consists of:

1. a set of vertices V = $A \cup P \cup AG$, where A is a set of artifacts, P is a set of processes, and AG is a set of agents;

2. a set of edges E = $E_u \cup E_g \cup E_d \cup E_t \cup E_c$, where i) $E_u \subseteq P \times A$ and $(p, a) \in E_u$ states that process p *used* artifact a, ii) $E_g \subseteq A \times P$ and $(a, p) \in E_g$ states that artifact a *was generated by* process p, iii) $E_d \subseteq A \times A$ and $(a_1, a_2) \in E_d$ states that artifact a_1 *was derived from* artifact a_2, iv) $E_t \subseteq P \times P$ and $(p_1, p_2) \in E_t$ states that process p_1 *was triggered by* process p_2, and v) $E_c \subseteq P \times AG$ and $(p, ag) \in E_c$ states that process p *was controlled by* agent ag.

In particular, we use a *tuple*, a list of name and value pairs, to denote the properties of nodes and edges in an OPM graph. Figure 5.2 shows a sample OPM graph that represents dependencies associated with process p_2 in Figure 5.1.

graph OG {
  node v_1<id='a_2', value=''>;
  node v_2<id='a_3', value=''>;
  node v_3<id='p_1', value='IsCSVReadyFileExists'>;
  node v_4<id='p_2', value='ReadCSVReadyFile'>;
  node v_5<id='ag_1', value='John'>;
  edge e_1(v_4,v_1)<id='u_2', role='used'>;
  edge e_2(v_2,v_4)<id='g_2', role='wasGeneratedBy'>;
  edge e_3(v_4,v_3)<id='t_1', role='wasTriggeredBy'>;
  edge e_4(v_4,v_5)<id='c_2', role='wasControlledBy'>;
};

Figure 5.2: A sample OPM graph representing dependencies associated with process p_2.

www.manaraa.com

5.2.2 Graph Patterns

We extend the notion of graph pattern proposed in [98] to efficiently support provenance queries over an OPM graph. In this work, we define six types of graph patterns, which are the main building blocks of an *OPQL* query.

Definition 5.2.1 (Graph Pattern: Type $\mathbb{B}$) A *graph pattern* P_b is a pair (M, C), where M is a graph motif and C is a predicate on the properties of the motif. Figure 5.3 shows a sample graph pattern of P_b.

graph P_b {
  node v_1;
  node v_2;
}
where v_1.value = 'butter'
and v_2.value = 'bake';

Figure 5.3: A sample graph pattern of P_b.

Definition 5.2.2 (Graph Pattern: Type $\mathbb{O}$) A *graph pattern* P_o is a triple (M, O, C), where M is a graph motif, O is an inverse-functional one-to-many mapping that returns a set of nodes that have direct causal dependencies associated with a node, and C is a predicate on the properties of the motif. To efficiently handle five causal dependencies between nodes defined in the OPM model [18], O is composed of ten types of mapping functions (i.e., $O \in \{O_u, O_{\hat{u}}, O_g, O_{\hat{g}}, O_d, O_{\hat{d}}, O_t, O_{\hat{t}}, O_c, O_{\hat{c}}\}$)

www.manaraa.com

as defined below:

$$
\begin{aligned}
O_u(p) &= \{a \mid (p,a) \in E_u\} \\
O_{\hat{u}}(a) &= \{p \mid (p,a) \in E_u\} \\
O_g(a) &= \{p \mid (a,p) \in E_g\} \\
O_{\hat{g}}(p) &= \{a \mid (a,p) \in E_g\} \\
O_d(a_1) &= \{a_2 \mid (a_1,a_2) \in E_d\} \\
O_{\hat{d}}(a_2) &= \{a_1 \mid (a_1,a_2) \in E_d\} \\
O_t(p_1) &= \{p_2 \mid (p_1,p_2) \in E_t\} \\
O_{\hat{t}}(p_2) &= \{p_1 \mid (p_1,p_2) \in E_t\} \\
O_c(p) &= \{ag \mid (p,ag) \in E_c\} \\
O_{\hat{c}}(ag) &= \{p \mid (p,ag) \in E_c\}
\end{aligned}
\tag{5.1}
$$

Graph pattern P_o is a derived graph pattern. It enables users to efficiently formulate complicated provenance queries. Figure 5.4 shows a sample graph pattern of P_o. In Figure 5.4, the former (graph pattern P_o) is derived by the latter (graph pattern P_b).

Next, we define the following four graph patterns to efficiently support tracking of ancestor nodes.

Definition 5.2.3 (Graph Pattern: Type $\mathbb{D}$) A *graph pattern* P_d is a triple (M, D, C), where M is a graph motif, D is an inverse-functional one-to-many mapping that returns a set of artifacts that were applied to derive an artifact, and C is a predicate on the properties of the motif. D is defined as:

$$
D(a) = \bigcup_{a' \in O_d(a)} D(a') \ \cup \ O_d(a) \tag{5.2}
$$

Graph pattern P_d is a derived graph pattern. It enables users to efficiently formulate recursive queries to track ancestor nodes regarding artifacts. For example, Figure 5.5 shows a sample graph

www.manaraa.com

graph P_o {
  node v_1;
  node v_2;
}
mapping $O_u : v_1 \xrightarrow{used} v_2$
where v_1.id = 'p_1';

(is derived by)

graph P_b {
  node v_1;
  node v_2;
}
where $e_1(v_1, v_2)$.role = 'used'
and v_1.id = 'p_1';

Figure 5.4: A sample graph pattern of P_o.

pattern of P_d. In Figure 5.5, the former (graph pattern P_d) is derived by the latter (graph pattern P_b) via the recursive graph pattern of P_b, which is represented in the ***with ~ union all*** clause.

Definition 5.2.4 (Graph Pattern: Type $\mathbb{T}$) A *graph pattern* P_t is a triple (M, T, C), where M is a graph motif, T is an inverse-functional one-to-many mapping that returns a set of processes that were applied to trigger a process, and C is a predicate on the properties of the motif. T is defined as:

$$T(p) = \bigcup_{p' \in O_t(p)} T(p') \ \cup \ O_t(p) \tag{5.3}$$

Graph pattern P_t is also a derived graph pattern. It enables users to efficiently formulate recursive queries to track ancestor nodes regarding processes. Figure 5.6 shows a sample graph pattern of P_t, where the former (graph pattern P_t) is derived by the latter (graph pattern P_b) via the recursive graph pattern of P_b, which is represented in the ***with ~ union all*** clause.

Definition 5.2.5 (Graph Pattern: Type $\mathbb{G}$) A *graph pattern* P_g is a triple (M, G, C), where M is a graph motif, G is an inverse-functional one-to-many mapping that returns a set of processes that

www.manaraa.com

graph P_d {
  node v_1;
  node v_2;
}
mapping $D : v_1 \xrightarrow{wasDerivedFrom^*} v_2$
where v_1.id = 'a_n';

(is derived by)

with graph P_b **as** A {
  node $A.v_1$;
  node $A.v_2$;
  }
  where $A.e_1(A.v_1, A.v_2)$.role = 'wasDerivedFrom'
  and $A.v_1$.id = 'a_n';
union all
  graph P_b **as** R {
    node $R.v_1$;
    node $R.v_2$;
  }
  where $R.e_1(R.v_1, R.v_2)$.role = 'wasDerivedFrom'
  and $R.v_1$.id = $A.v_2$.id;

Figure 5.5: A sample graph pattern of P_d.

were applied to generate an artifact, and C is a predicate on the properties of the motif. G is defined as:

$$G(a) = \bigcup_{p' \in O_g(a)} T(p') \ \cup \ O_g(a) \tag{5.4}$$

Definition 5.2.6 (Graph Pattern: Type $\mathbb{U}$) A *graph pattern* P_u is a triple (M, U, C), where M is a graph motif, U is an inverse-functional one-to-many mapping that returns a set of artifacts that were used by a process, and C is a predicate on the properties of the motif. U is defined as:

$$U(p) = \bigcup_{a' \in O_u(p)} D(a') \ \cup \ O_u(p) \tag{5.5}$$

www.manaraa.com

graph P_t {
  node v_1;
  node v_2;
}
mapping $T : v_1 \xrightarrow{wasTriggeredBy^*} v_2$
where v_1.id = 'p_m';

(is derived by)

with graph P_b **as** A {
  node $A.v_1$;
  node $A.v_2$;
  }
  where $A.e_1(A.v_1, A.v_2)$.role = 'wasTriggeredBy'
  and $A.v_1$.id = 'p_m';
union all
  graph P_b **as** R {
    node $R.v_1$;
    node $R.v_2$;
  }
  where $R.e_1(R.v_1, R.v_2)$.role = 'wasTriggeredBy'
  and $R.v_1$.id = $A.v_2$.id;

Figure 5.6: A sample graph pattern of P_t.

As described above, graph patterns P_g and P_u are derived graph patterns. These graph patterns enable users to efficiently formulate recursive queries to track ancestor nodes regarding processes and artifacts, respectively. In a similar fashion, sample graph patterns of P_g and P_u can be described.

Next, we define three types of *graph pattern matching* which generalize subgraph isomorphism over six graph patterns.

Definition 5.2.7 (Graph Pattern Matching** α**) A *graph pattern* P_b is matched with a graph OG if there exists an injective mapping ϕ_α: $V(M) \rightarrow V(OG)$ such that i) For $\forall\ e(u, v) \in E(M)$, $(\phi_\alpha(u), \phi_\alpha(v))$ is an edge in OG, and ii) predicate $C_{\phi_\alpha}(OG)$ holds.

www.manaraa.com

Definition 5.2.8 (Graph Pattern Matching β) A *graph pattern* P_o is matched with a graph OG if there exists an injective mapping ϕ_β: $V(M) \rightarrow V(OG)$ such that i) For $\forall\ e(u,v) \in E(M)$, $(\phi_\beta(u), \phi_\beta(v))$ is an edge in OG, ii) function $O_{\phi_\beta}(OG)$ holds, and iii) predicate $C_{\phi_\beta}(OG)$ holds.

Definition 5.2.9 (Graph Pattern Matching γ) Each of graph patterns (P_d, P_t, P_g, and P_u) is matched with a graph OG if there exists an injective mapping ϕ_γ: $V(M) \rightarrow V(OG)$ such that i) For $\forall\ e(u,v) \in E(M)$, $(\phi_\gamma(u), \phi_\gamma(v))$ is an edge in OG, ii) each function ($D_{\phi_\gamma}(OG)$, $T_{\phi_\gamma}(OG)$, $G_{\phi_\gamma}(OG)$, and $U_{\phi_\gamma}(OG)$) holds, and iii) each predicate $C_{\phi_\gamma}(OG)$ holds.

To denote the binding between a graph pattern and an OPM graph, we define a *matched graph* as follows.

Definition 5.2.10 (Matched Graph) Given an injective mapping $\phi \in \{\phi_\alpha, \phi_\beta, \phi_\gamma\}$ between a pattern $P \in \{P_b, P_o, P_d, P_t, P_g, P_u\}$ and an OPM graph OG, a matched graph is a triple (ϕ, P, OG) and is defined as $\phi_P(OG)$.

5.2.3 OPM-Based Graph Algebra

We propose an OPM-based graph algebra for the *OPQL* query language. The OPM-based graph algebra is based on four operators, which operate on an OPM graph. Each operator takes one OPM graph as input and produces another OPM graph as output. In particular, each of the union, intersection, and difference operators is basically operated on one OPM graph, but these operators take two OPM subgraphs produced by other queries as input and produce an OPM graph as output. We define the following four operators to manipulate and query an OPM graph.

5.3.3.1 Extract operator (δ)

One of the most frequent operations performed on an OPM graph is the extraction of a set of nodes and edges, which are constituents of an OPM graph. An extract operator is defined using a graph pattern P. It takes one OPM graph (OG) as input and produces a new OPM graph that matches the graph pattern as output, denoted by $\delta_P(OG)$. For example, let Figure 5.1 be an OPM graph (OG).

www.manaraa.com

You might want to find all the artifacts that contributed to derive artifact a_6. Using the extract operator, this query can be expressed as:

$$\delta_{[P_d:D(a_6)]}(OG) \tag{5.6}$$

This query first generalizes a matched graph which consists of a set of artifacts (a_1-a_6) and a set of edges (d_1-d_5) via the *graph pattern matching* γ (i.e., ϕ_γ), and then it produces a new OPM graph by combining information from the matched graph. The output of the extract operator is an OPM graph:

$$\delta_p(OG) = \phi_P(OG) \tag{5.7}$$

Next, we define the following three operators (*union*, *intersection*, and *difference*). These operators are basically operated on one OPM graph, but they take two OPM subgraphs produced by other queries as input and produce an OPM graph as output. Let OG be an OPM graph, and let OG_1 and OG_2 be the output of $\delta_{P_1}(OG)$ and $\delta_{P_2}(OG)$, respectively. Given two OPM subgraphs $OG_1 = (V_1, E_1)$ and $OG_2 = (V_2, E_2)$, where OG_1 and $OG_2 \subseteq OG$, these operators are defined as follows.

5.3.3.2 Union operator ($\cup$)

The union operator calculates the union of two OPM subgraphs. A union operation is defined by $OG_1 \cup OG_2$, resulting in an OPM graph $OG' = (V', E')$, where

$$\begin{aligned} V' &= \{v \mid v \in V_1 \ \ or \ \ v \in V_2\} \\ E' &= \{e \mid e \in E_1 \ \ or \ \ e \in E_2\} \end{aligned} \tag{5.8}$$

For example, let Figure 5.7(a) be an OPM graph (OG). Then, Figure 5.7(b) and Figure 5.7(c) represent the output of $\delta_{[P_d:D(a_5)]}(OG)$ and $\delta_{[P_d:D(a_8)]}(OG)$, respectively. You might want to find all the artifacts that contributed to derive either artifact a_5 or artifact a_8 over OPM graph OG. Using

www.manaraa.com

the union operator, this query can be expressed as $\delta_{[P_d:D(a_5)]}(OG) \cup \delta_{[P_d:D(a_8)]}(OG)$. The result of the query is shown in Figure 5.7(d).

5.3.3.3 Intersection operator ($\cap$)

The intersection operator calculates the intersection of two OPM subgraphs. An intersection operation is defined by $OG_1 \cap OG_2$, resulting in an OPM graph OG' = (V', E'), where

$$\begin{aligned} V' &= \{v \mid v \in V_1 \ \ and \ \ v \in V_2\} \\ E' &= \{e \mid e \in E_1 \ \ and \ \ e \in E_2\} \end{aligned} \tag{5.9}$$

For example, you might want to find all the artifacts that contributed to derive both artifact a_5 and artifact a_8 over OPM graph OG. Using the intersection operator, this query can be expressed as $\delta_{[P_d:D(a_5)]}(OG) \cap \delta_{[P_d:D(a_8)]}(OG)$. The result of the query is shown in Figure 5.7(e).

5.3.3.4 Difference operator ($-$)

The difference operator calculates the difference of two OPM subgraphs. A difference operation is defined by $OG_1 - OG_2$, resulting in an OPM graph OG' = (V', E'), where

$$\begin{aligned} V' &= \{v \mid v \in V_1 \ \ and \ \ v \notin V_2\} \\ E' &= \{e \mid e \in E_1 \ \ and \ \ e \notin E_2\} \end{aligned} \tag{5.10}$$

For example, you might want to find all the artifacts that contributed to derive artifact a_5, but not artifact a_3 over OPM graph OG. Using the difference operator, this query can be expressed as $\delta_{[P_d:D(a_5)]}(OG) - \delta_{[P_d:D(a_3)]}(OG)$. The result of the query is shown in Figure 5.7(f).

5.3.3.5 Example Provenance Queries Expressed using the OPM-based Graph Algebra

To evaluate the feasibility of the operators defined in the OPM-based graph algebra, we use eight example provenance queries, which require the computation of transitive relationships to track ancestor nodes. These queries, including four queries (Q1-Q4) for the Load Workflow defined in the

www.manaraa.com

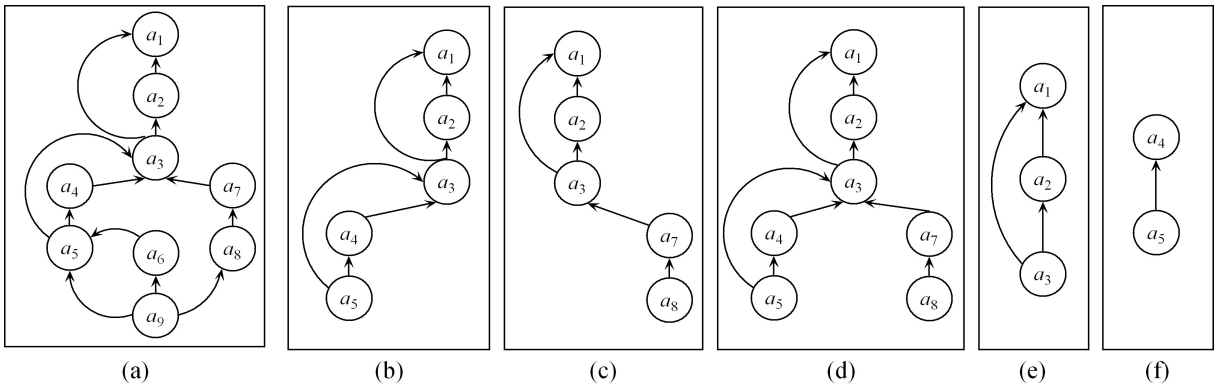


Figure 5.7: The output produced by the operation of different operators.

Third Provenance Challenge [15] and four queries (Q5-Q8) for a synthetic workflow consisting of a large number of steps, can be expressed using our OPM-based graph algebra (these queries can be also expressed in *OPQL* as shown later in Figure 5.14). First, let OG_1 and OG_2 be the OPM graphs produced by the execution of the Load Workflow and synthetic workflow, respectively. Then, as depicted in Table 5.1, query Q1, which asks for CSV files that contributed to a given detection, can be answered by a query expressed as $\delta_{[P_d:D('detectID')]}(OG_1) \cap \delta_{[P_b:value='\%CSV\%']}(OG_1)$. It first finds all the artifacts that contributed to derive the artifact with the value "detectID" by $\delta_{[P_d:D('detectID')]}(OG_1)$, and then it retrieves these artifacts whose value is CSV files via the intersection with $\delta_{[P_b:value='\%CSV\%']}(OG_1)$. Similarly, query Q5 can be answered by $\delta_{[P_d:D(a_n)]}(OG_2)$. Second, query Q2, which asks for steps that were completed successfully before the halt occurred, can be answered by a query expressed as $\delta_{[P_g:G('\%success\%')]}(OG_1)$ to find all the processes that contributed to generate artifacts with the value "%success%". In a similar fashion, the answers of queries Q3, Q4, and Q8 can be expressed as depicted in Table 5.1. Finally, query Q6, which asks for a process dependency view for all the steps that contributed to trigger the last step (id = p_n), can be satisfied by using $\delta_{[P_t:T(p_n)]}(OG_2)$ and query Q7, which asks for a data dependency view for all the data products that were directly or indirectly used by the last step (id = p_n), can be satisfied by using $\delta_{[P_u:U(p_n)]}(OG_2)$.

www.manaraa.com

Table 5.1: The provenance queries expressed using the OPM-based graph algebra.

Q1:	For a given detection (detectID), which CSV files contributed to it? $\Rightarrow \delta_{[P_d:D('detectID')]}(OG_1) \cap \delta_{[P_b:value='\%CSV\%']}(OG_1)$
Q2:	Which steps were completed successfully before the halt occurred? $\Rightarrow \delta_{[P_g:G('\%success\%')]}(OG_1)$
Q3:	Why is this entry (ccdID) in the database? $\Rightarrow \delta_{[P_g:G('ccdID')]}(OG_1)$
Q4:	Which operation executions were necessary for the Image table to contain a particular value? $\Rightarrow \delta_{[P_g:G('\%Image\%')]}(OG_1)$
Q5:	Display dependencies of all the data products that contributed to derive the last data product (id = a_n). $\Rightarrow \delta_{[P_d:D(a_n)]}(OG_2)$
Q6:	Display dependencies of all the steps that were applied to trigger the last step (id = p_n). $\Rightarrow \delta_{[P_t:T(p_n)]}(OG_2)$
Q7:	Display dependencies of all the data products that were used by the last step (id = p_n). $\Rightarrow \delta_{[P_u:U(p_n)]}(OG_2)$
Q8:	Display dependencies of all the steps that contributed to generate the last data product (id = a_n). $\Rightarrow \delta_{[P_g:G(a_n)]}(OG_2)$

5.2.4 OPQL Syntax and Semantics

We present an *OPQL* syntax that is required to formulate *OPQL* queries and a formal semantics for *OPQL* constructs. *OPQL* queries are formulated against an OPM graph displayed by a graphical user interface.

5.3.4.1 OPQL Syntax

OPQL queries are built from the following syntax as depicted in Figure 5.8. An *OPQL* query is composed of either a basic query or a set operation of two queries via set operators (i.e., UNION, INTERSECT, and MINUS). A basic query can be one of the single-node constructs (A, P, and AG), one of the single-step-edge-forward constructs (USD, WGB, WCB, WDF, and WTB), one of the single-step-edge-backward constructs ($USD^{\wedge}$, $WGB^{\wedge}$, $WCB^{\wedge}$, $WDF^{\wedge}$, and $WTB^{\wedge}$), or one of the multi-step-edge constructs (USD*, WGB*, WDF*, and WTB*). Each of these constructs has an argument (arg) in a bracket, and an argument for a construct can be either a node expression (X_n) or a basic query. If a construct has a basic query as an argument, it means a nested *OPQL* query; otherwise, it means a simple *OPQL* query. A node expression (X_n) can be expressed by an artifact node expression (X_a), a process node expression (X_p), or an agent node expression (X_{ag}).

www.manaraa.com

query :: = basic-query
| query UNION query
| query INTERSECT query
| query MINUS query
basic-query :: = single-node construct (arg)
| single-step-edge-forward construct (arg)
| single-step-edge-backward construct (arg)
| multi-step-edge construct (arg)
single-node construct :: = A | P | AG
single-step-edge-forward construct :: =
USD | WGB | WCB | WDF | WTB
single-step-edge-backward construct :: =
$USD^{\wedge} \mid WGB^{\wedge} \mid WCB^{\wedge} \mid WDF^{\wedge} \mid WTB^{\wedge}$
multi-step-edge construct :: =
USD* | WGB* | WDF* | WTB*
arg :: = basic-query | node-expression (X_n)
node-expression (X_n) :: = artifact-node-expression (X_a)
| process-node-expression (X_p)
| agent-node-expression (X_{ag})
artifact-node-expression (X_a) :: =
artifact-identifier (a_n) | %artifact-value% (a_ν) | a^*
process-node-expression (X_p) :: =
process-identifier (p_n) | %process-value% (p_ν) | p^*
agent-node-expression (X_{ag}) :: =
agent-identifier (ag_n) | %agent-value% (ag_ν) | ag^*

Figure 5.8: The *OPQL* Syntax.

The node expressions of an artifact, process, and agent can be a node identifier (i.e., a_n, p_n, or ag_n), a node value (i.e., a_ν, p_ν, or ag_ν) starting and ending with % , or a wildcard (i.e., a^*, p^*, or ag^*), respectively.

5.3.4.2 OPQL Semantics

Let $OG = (V, E)$ be an OPM graph such that $V = A \cup P \cup AG$ and $E = E_u \cup E_g \cup E_c \cup E_d \cup E_t$, as defined in Section 3.1. First, each node expression X_n (i.e., $X_n \in \{X_a, X_p, X_{ag}\}$) is defined as a function that maps an OPM graph (OG) to a set of vertices such that $X_n(OG)$ returns a subset of

www.manaraa.com

$$
\begin{aligned}
\|a_n\| &= \{a_n\} \\
\|p_n\| &= \{p_n\} \\
\|ag_n\| &= \{ag_n\} \\
\|a_\nu\| &= \{a_n \mid a_n \in ids(a_\nu) \ \ and \ \ a_n \in A\} \\
\|p_\nu\| &= \{p_n \mid p_n \in ids(p_\nu) \ \ and \ \ p_n \in P\} \\
\|ag_\nu\| &= \{ag_n \mid ag_n \in ids(ag_\nu) \ \ and \ \ ag_n \in AG\} \\
\|a^*\| &= \{a_n \mid a_n \in A\} \\
\|p^*\| &= \{p_n \mid p_n \in P\} \\
\|ag^*\| &= \{ag_n \mid ag_n \in AG\}
\end{aligned}
$$

Figure 5.9: The semantics of node expression X_n.

$$
\begin{aligned}
A(X_a) &= \{a_n \mid a_n \in X_a\} \\
P(X_p) &= \{p_n \mid p_n \in X_p\} \\
AG(X_{ag}) &= \{ag_n \mid ag_n \in X_{ag}\} \\
USD(X_p) &= \{a_n \mid p_n \in X_p \ \ and \ \ (p_n, a_n) \in E_u\} \\
WGB(X_a) &= \{p_n \mid a_n \in X_a \ \ and \ \ (a_n, p_n) \in E_g\} \\
WCB(X_p) &= \{ag_n \mid p_n \in X_p \ \ and \ \ (p_n, ag_n) \in E_c\} \\
WDF(X_a) &= \{a_{n2} \mid a_{n_1} \in X_a \ \ and \ \ (a_{n_1}, a_{n_2}) \in E_d\} \\
WTB(X_p) &= \{p_{n2} \mid p_{n_1} \in X_p \ \ and \ \ (p_{n_1}, p_{n_2}) \in E_t\} \\
USD^{\wedge}(X_a) &= \{p_n \mid a_n \in X_a \ \ and \ \ (p_n, a_n) \in E_u\} \\
WGB^{\wedge}(X_p) &= \{a_n \mid p_n \in X_p \ \ and \ \ (a_n, p_n) \in E_g\} \\
WCB^{\wedge}(X_{ag}) &= \{p_n \mid ag_n \in X_{ag} \ \ and \ \ (p_n, ag_n) \in E_c\} \\
WDF^{\wedge}(X_a) &= \{a_{n_1} \mid a_{n_2} \in X_a \ \ and \ \ (a_{n_1}, a_{n_2}) \in E_d\} \\
WTB^{\wedge}(X_p) &= \{p_{n_1} \mid p_{n_2} \in X_p \ \ and \ \ (p_{n_1}, p_{n_2}) \in E_t\}
\end{aligned}
$$

Figure 5.10: The semantics of the single-node and single-step-edge-forward (backward) constructs.

V as depicted in Figure 5.9, where $ids(n_\nu)$ returns those nodes satisfying node value $n_\nu \in \{a_\nu, p_\nu, ag_\nu\}$.

We define the following three types of *OPQL* constructs including single-node constructs, single-step-edge-forward constructs, and single-step-edge-backward constructs. First, the single-node constructs play a role to efficiently retrieve nodes in an OPM graph, and they are defined as functions that take an OPM graph $OG = (V, E)$ and a node expression X_n and return those

www.manaraa.com

$$WDF^*(X_a) = \{a_n \mid \bigcup_{a_n \in WDF(X_a)} WDF^*(a_n) \ \cup \ WDF(X_a)\}$$
$$WTB^*(X_p) = \{p_n \mid \bigcup_{p_n \in WTB(X_p)} WTB^*(p_n) \ \cup \ WTB(X_p)\}$$
$$WGB^*(X_a) = \{p_n \mid \bigcup_{p_n \in WGB(X_a)} WTB^*(p_n) \ \cup \ WGB(X_a)\}$$
$$USD^*(X_p) = \{a_n \mid \bigcup_{a_n \in USD(X_p)} WDF^*(a_n) \ \cup \ USD(X_p)\}$$

Figure 5.11: The semantics of the multi-step-edge constructs.

nodes satisfying node expression X_n such that $X_n(OG) \subseteq V$. Specifically, given single-node construct C_n (i.e., $C_n \in \{A, P, AG\}$), OPM graph $OG = (V, E)$, and node expression X_n, the semantics of the single-node constructs are defined by $C_n(X_n, OG) = \{n \mid n \in X_n(OG) \subseteq V\}$. For convenience, we generally omit OG when writing ***OPQL*** constructs and node expressions (as in Figure 5.9, 5.10, 5.11, 5.12, and 5.14). Second, the single-step-edge-forward constructs and single-step-edge-backward constructs play a role to efficiently retrieve the cause node (the destination of an arc) and the effect node (the source of an arc) representing a causal dependency between two nodes in an OPM graph, respectively. The single-step-edge-forward constructs are defined as functions that take an OPM graph $OG = (V, E)$ and a node expression X_n for effect nodes and return cause nodes which have causal dependencies with effect nodes satisfying $X_n(OG)$, while the single-step-edge-backward constructs are defined as functions that take an OPM graph $OG = (V, E)$ and a node expression X_n for cause nodes and return effect nodes which have causal dependencies with cause nodes satisfying $X_n(OG)$. Specifically, given single-step-edge-forward construct C_e (i.e., $C_e \in \{USD, WGB, WCB, WDF, WTB\}$), single-step-edge-backward construct $C_{\hat{e}}$ (i.e., $C_{\hat{e}} \in \{USD^\wedge, WGB^\wedge, WCB^\wedge, WDF^\wedge, WTB^\wedge\}$), OPM graph $OG = (V, E)$, and node expression X_n, the semantics of the single-step-edge-forward constructs and single-step-edge-backward constructs are defined by $C_e(X_n, OG) = \{n^{cause} \mid n^{effect} \in X_n \ and \ (n^{effect}, n^{cause}) \in E\}$ and $C_{\hat{e}}(X_n, OG) = \{n^{effect} \mid n^{cause} \in X_n \ and \ (n^{effect}, n^{cause}) \in E\}$, respectively. More details on the semantics of these constructs are shown in Figure 5.10.

www.manaraa.com

Next, we define the multi-step-edge constructs (i.e., WDF^*, WTB^*, WGB^*, and USD^*) as functions that take an OPM graph $OG = (V, E)$ and a node expression X_n and return all the nodes which have direct or indirect causal dependencies (i.e., transitive relationships) with those nodes satisfying $X_n(OG)$. These constructs allow user to efficiently track ancestor nodes without formulating recursive queries. For example, let Figure 5.7(a) be an OPM graph (OG) as input. Then, multi-step-edge construct $WDF^*(a_5)$ returns all the artifacts that contributed to derive artifact a_5. That is, it returns a set of artifacts $\{a_1, a_2, a_3, a_4\}$ by the computation of transitive relationships associated with artifact a_5 via existing causal dependencies (i.e., $\{(a_5, a_4), (a_5, a_3), (a_4, a_3), (a_3, a_2), (a_3, a_1), (a_2, a_1)\} \subset E_d$). The semantics of these multi-step-edge constructs are depicted in Figure 5.11.

A simple *OPQL* query is formulated by only an *OPQL* construct which has a node expression as an argument. Then, through graph pattern matching over the nodes returned by the computation of a single construct, a new OPM graph as output is extracted. In a similar way, a nested *OPQL* query is formulated by a combination of the *OPQL* constructs, and a new OPM graph as output is extracted via graph pattern matching over all the nodes that were returned by the computation of all the constructs in a nested *OPQL* query. To give a better understanding, we present simple *OPQL* query examples in Figure 5.12, where the description of the *OPQL* constructs and the graphical query results are described.

In addition, an *OPQL* query can be expressed using a set operator between two *OPQL* queries since the result of a basic query is an OPM graph which consists of a set of vertices and a set of edges. For example, given two *OPQL* queries Q_1 and Q_2, a new *OPQL* query combining these two queries can be formulated using set operators UNION, INTERSECT, and MINUS (e.g., Q_1 UNION Q_2, Q_1 INTERSECT Q_2, and Q_1 MINUS Q_2). More details on the *OPQL* query expression are discussed in the following section.

www.manaraa.com

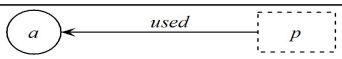
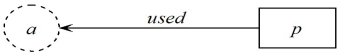
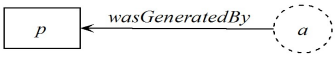
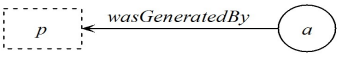
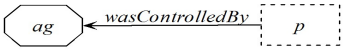
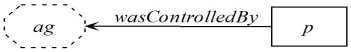
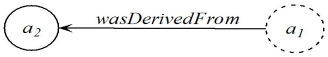
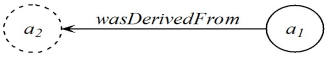
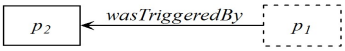
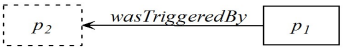
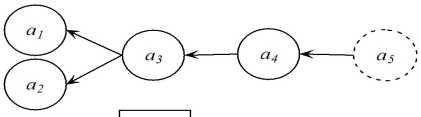
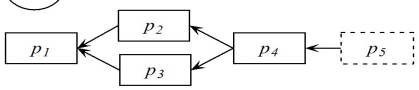
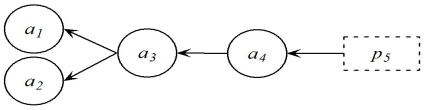
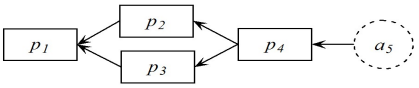
Construct	Description	Graphical Query Result
A (a)	Find artifacts satisfying artifact a	
P (p)	Find processes satisfying process p	
AG (ag)	Find agents satisfying agent ag	
USD (p)	Find artifacts that process p used	
USD^ (a)	Find processes that used artifact a	
WGB (a)	Find processes that generated artifact a	
WGB^ (p)	Find artifacts that process p generated	
WCB (p)	Find agents that controlled process p	
WCB^ (ag)	Find processes that agent ag controlled	
WDF (a_1)	Find artifacts that derived artifact a_1	
WDF^ (a_2)	Find artifacts that artifact a_2 derived	
WTB (p_1)	Find processes that triggered process p_1	
WTB^ (p_2)	Find processes that process p_2 triggered	
WDF* (a_5)	Find all the artifacts that were applied to derive artifact a_5	
WTB* (p_5)	Find all the processes that were applied to trigger process p_5	
USD* (p_5)	Find all the artifacts that process p_5 used directly or indirectly	
WGB* (a_5)	Find all the processes that were applied to generate artifact a_5	

Figure 5.12: The description of the *OPQL* constructs and the graphical query results.

5.2.5 *Expressing Provenance Queries in OPQL*

We discuss how provenance queries can be expressed in *OPQL*. As described in Section 3.4, an *OPQL* query is expressed as a combination of *OPQL* constructs, each of which corresponds to each of the graph patterns. The *OPQL* query language provides users with effective query formulation. For example, Figure 5.13 shows two different query expressions that generate a data dependency graph (*DG*) for artifact a_6 over the OPM graph depicted in Figure 5.1. First, Figure 5.13(a) shows

www.manaraa.com

Given OPM graph OG,
DG = **WDF*** (a_6);

(a)

with graph P_b **as** A {
  node $A.v_1$;
  node $A.v_2$;
  }
  where $A.e_1(A.v_1, A.v_2)$.role = 'wasDerivedFrom'
  and $A.v_1$.id = 'a_6';
union all
  graph P_b **as** R {
    node $R.v_1$;
    node $R.v_2$;
  }
  where $R.e_1(R.v_1, R.v_2)$.role = 'wasDerivedFrom'
  and $R.v_1$.id = $A.v_2$.id;

DG = **graph** { };
for A in doc (OG)
let DG: = **graph** {
  graph DG;
  node $A.v_1$, $A.v_2$;
  edge $A.e_1(A.v_1, A.v_2)$;
  unify $DG.v_2$, $A.v_1$ **where** $DG.v_2.id$ = $A.v_1$;
}

(b)

Figure 5.13: Two different query expressions that generate a data dependency graph (DG).

an *OPQL* query expression to answer the query via an *OPQL* construct, and then Figure 5.13(b) shows a GraphQL query expression [98], which is expressed by a graph pattern and a FLWR (*For, Let, Where, Return*) expression in XQuery. Although the query expressed in GraphQL results in the same output as that of the *OPQL* query, the GraphQL query requires users to directly write a recursive query with a graph pattern; on the other hand, *OPQL* allows users to effectively formulate the query with just writing *WDF**(a_6). *OPQL* supports graph queries at a higher level than GraphQL.

www.manaraa.com

In addition, to demonstrate the expressiveness of *OPQL*, we use 16 provenance queries, including three core queries (CQ) and 13 optional queries (OQ) defined in the Third Provenance Challenge [15]. In particular, we express some of these queries in *OPQL* that are executable over our OPMPROV system. As depicted in Figure 5.14 (which is extended from a figure presented in [82], where these queries are presented in SQL), we present 13 provenance queries in English, SQL, and OPQL, respectively. In this dissertation, we omit the description of these queries and their answers (more details on these query processing can be found in [82]). Instead, we introduce an example about how a provenance query is formulated and answered by SQL and OPQL, respectively. For query CQ1, which asks for CSV files that contributed to a given detection, SQL uses relation *MultiStepWasGeneratedBy* to find all process identifiers contributed to the generation of the artifact with the value "261887437010025730 (detectID)" and then it retrieves artifacts that are CSV files used by those processes. On the other hand, *OPQL* uses construct *WGB** to find all processes that were contributed to generate an artifact whose value is "261887437010025730 (detectID)", and then it uses construct *USD* to find artifacts used by those processes, and then it intersects with construct *A(%Detection.csv%)* to retrieve artifacts that are CSV files. In terms of usability, *OPQL* supports more effective query formulation than SQL; furthermore, as depicted in Figure 5.15, a query result of an *OPQL* query is displayed as an OPM graph to give a better understanding to users, while an SQL query result is a set of tuples (i.e., a table).

5.3 Experimental Study

In this section, we report on our experimental study that explored the performance of OPMPROV on *OPQL* provenance querying over various datasets. Before we conduct experiments, we implement *OPQL* as a Web service via the OPMPROV system (we discuss the implementation of *OPQL* in the following chapter). We first performed provenance query experiments for *OPQL*, and then we performed provenance visualization performance experiments to demonstrate the visualization capability of OPMPROV. The experiments presented below were conducted on a PC

www.manaraa.com

CQ1: For a given detection, which CSV files contributed to it?
SQL:
```
SELECT DISTINCT A2.Value FROM Artifact A2, Used U,
 (SELECT DISTINCT TG.OPMGraphId, TG.ProcessId FROM MultiStepWasGeneratedBy TG, Artifact A WHERE TG.OPMGraphId = A.OPMGraphId AND
TG.ArtifactId = A.ArtifactId AND A.Value = '261887437010025730' ) As Pv WHERE U.OPMGraphId = Pv.OPMGraphId AND U.ProcessId = Pv.ProcessId AND
U.OPMGraphId = A2.OPMGraphId AND U.ArtifactId = A2.ArtifactId AND A2.Value LIKE '%Detection.csv' AND U.OPMGraphId = '1'
```
OPQL: **USD (WGB* (%261887437010025730%)) INTERSECT A (%Detection.csv%)**

CQ2: The user considers a table to contain values they do not expect. Was the range check (IsMatchTableColumnRanges) performed for this table?
SQL:
```
SELECT 'YES' AS Answer
FROM (SELECT COUNT(G.ArtifactId) AS Number FROM WasGeneratedBy G
WHERE G.ProcessId LIKE '%IsMatchTableColumnRanges%' AND G.OPMGraphId = '1') AS Output WHERE Output.Number > 0
```
OPQL: **WGB^ (%IsMatchTableColumnRanges%)**

CQ3: Which operation executions were strictly necessary for the Image table to contain a particular (non-computed) value?
SQL:
```
SELECT Cp.Value As Operation, SUM(Cp.Number) As Count
FROM (SELECT DISTINCT TT.OPMGraphId, TT.CauseProcessId, P.Value, 1 As Number FROM MultiStepWasTriggeredBy TT, Artifact A, Process P,
WasGeneratedBy G WHERE TT.OPMGraphId = G.OPMGraphId AND G.OPMGraphId = A.OPMGraphId AND TT.EffectProcessId = G.ProcessId AND
G.ArtifactId = A.ArtifactId AND A.Value LIKE '%Image%' AND TT.OPMGraphId = P.OPMGraphId AND TT.CauseProcessId = P.ProcessId) As Cp WHERE
Cp.OPMGraphId = '1' GROUP BY Cp.Value
```
OPQL: **WTB* (WGB (%Image%)) INTERSECT P (%LoadCSVFileIntoTable%)**

OQ1: How many tables successfully loaded before the workflow halted due to a failed check?
SQL:
```
SELECT COUNT(*) AS Count FROM Artifact A, WasGeneratedBy G
WHERE A.OPMGraphId = G.OPMGraphId AND A.ArtifactId = G.ArtifactId AND G.ProcessId LIKE 'IsMatchTableColumnRanges%' AND A.Value LIKE
'%success%' AND A.OPMGraphId = '1'
```
OPQL: **WGB^ (%IsMatchTableColumnRanges%) INTERSECT A (%success%)**

OQ4: Why is this entry in the database?
SQL:
```
SELECT Pv.Value, SUM (Pv.Number) As Count FROM (SELECT DISTINCT TT.OPMGraphId, TT.CauseProcessId, P.Value, 1 As Number
FROM MultiStepWasTriggeredBy TT, Artifact A, WasGeneratedBy G, Process P WHERE TT.OPMGraphId = G.OPMGraphId AND TT.EffectProcessId =
G.ProcessId AND TT.OPMGraphId = P.OPMGraphId AND TT.CauseProcessId = P.ProcessId AND A.OPMGraphId = G.OPMGraphId AND A.ArtifactId =
G.ArtifactId AND A.Value = ' 261887437010025730') AS Pv WHERE Pv.OPMGraphId = '1' GROUP BY Pv.Value
```
OPQL: **WTB* (WGB (%261887437010025730%))**

OQ5: A user executes the workflow many times (say 5 times) over different sets of data (j062941, ..., j062945). He wants to determine, which of the execution halted?
SQL:
```
SELECT A.OPMGraphId, A.Value AS NameOfDataset
FROM Artifact A, WasGeneratedBy W WHERE A.OPMGraphId = W.OPMGraphId AND A.ArtifactId = W.ArtifactId AND A.Value LIKE 'J%halt%'
```
OPQL: **WGB (%J%halt%)**

OQ6: Determine the step where Halt occurred?
SQL:
```
SELECT Hp.Value As HaltStep, SUM(Hp.Number) As Count FROM (SELECT DISTINCT P.OPMGraphId, P.ProcessId, P.Value, 1 As Number
FROM Artifact A, WasGeneratedBy G, Process P WHERE A.Value LIKE '%halt%' AND A.OPMGraphId = G.OPMGraphId AND A.ArtifactId = G.ArtifactId AND
G.OPMGraphId = P.OPMGraphId AND G.ProcessId = P.ProcessId) As Hp WHERE Hp.OPMGraphId = '1' GROUP BY Hp.Value
```
OPQL: **WGB (%halt%)**

OQ7: Determine data and associated granularities of the data being processed, when halt occurred?
SQL:
```
SELECT DISTINCT A2.ArtifactId, A2.Value FROM Artifact A1, WasGeneratedBy G, Artifact A2, Used U
WHERE A1.Value LIKE '%halt%occurred%' AND A1.OPMGraphId = G.OPMGraphId AND A1.ArtifactId = G.ArtifactId AND U.OPMGraphId = G.OPMGraphId
AND U.ProcessId = G.ProcessId AND A2.OPMGraphId = U.OPMGraphId AND A2.ArtifactId = U.ArtifactId AND A2.OPMGraphId = '1'
```
OPQL: **USD (WGB (%halt%occurred%))**

OQ8: Which steps were completed successfully before the halt occurred?
SQL:
```
SELECT Sp.Value As Step, SUM (Sp.Number) As Count
FROM (SELECT DISTINCT TW.OPMGraphId, TW.ProcessId, P.Value, 1 As Number FROM MultiStepWasGeneratedBy TW, Artifact A, Process P WHERE
TW.OPMGraphId = A.OPMGraphId AND TW.ArtifactId = A.ArtifactId AND A.Value LIKE '%success%' AND TW.OPMGraphId = P.OPMGraphId AND
TW.ProcessId = P.ProcessId) As Sp WHERE Sp.OPMGraphId = '1' GROUP BY Sp.Value
```
OPQL: **WGB* (%success%)**

OQ10: For a workflow execution, determine the user inputs?
SQL:
```
SELECT A.Value FROM Used U, Artifact A WHERE U.OPMGraphId = A.OPMGraphId AND U.ArtifactId = A.ArtifactId AND NOT EXISTS (SELECT * FROM
WasGeneratedBy G WHERE G.OPMGraphId = A.OPMGraphId AND G.ArtifactId = A.ArtifactId) AND A.OPMGraphId = '1'
```
OPQL: **A (a*) MINUS WGB^ (p*)**

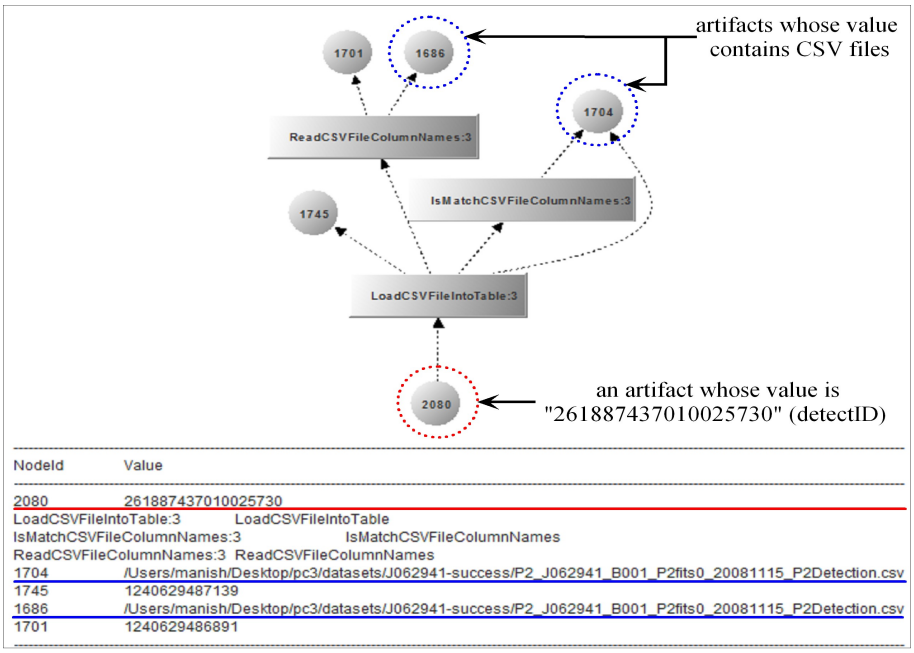
OQ11: For a workflow execution, determine steps that required user inputs?
SQL:
```
SELECT DISTINCT P.ProcessId, P.Value FROM Used U, Artifact A, Process P
WHERE U.OPMGraphId = A.OPMGraphId AND U.ArtifactId = A.ArtifactId AND U.OPMGraphId = P.OPMGraphId AND U.ProcessId = P.ProcessId AND NOT
EXISTS (SELECT * FROM WasGeneratedBy G WHERE G.OPMGraphId = A.OPMGraphId AND G.ArtifactId = A.ArtifactId) AND P.OPMGraphId = '1'
```
OPQL: **USD^ (A (a*)) MINUS WGB^ (p*)**

OQ12: For a workflow execution that halted, which files where processed successfully?
SQL:
```
SELECT DISTINCT A2.ArtifactId, A2.Value FROM WasGeneratedBy G, Artifact A1, Used U, Artifact A2 WHERE G.OPMGraphId = A1.OPMGraphId AND
G.ArtifactId = A1.ArtifactId AND A1.Value LIKE '%success%' AND U.OPMGraphId = G.OPMGraphId AND U.ProcessId = G.ProcessId AND A2.OPMGraphId =
U.OPMGraphId AND A2.ArtifactId = U.ArtifactId AND A2.Value LIKE '%CSVFileEntry%' AND A2.OPMGraphId = '1'
```
OPQL: **USD (WGB (%success%)) INTERSECT A (%CSVFileEntry%)**

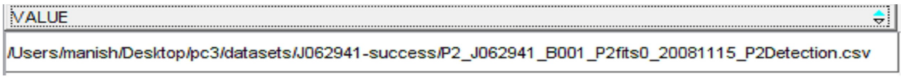
OQ13: Display the following provenance views: data dependency view and step dependency view.
SQL:
```
SELECT EffectProcessId, CauseProcessId FROM MultiStepWasTriggeredBy WHERE OPMGraphId = '1';
SELECT EffectArtifactId, CauseArtifactId FROM MultiStepWasDerivedFrom WHERE OPMGraphId = '1';
```
OPQL: **WDF* (a*); WTB* (p*)**

Figure 5.14: Provenance queries expressed by *OPQL* for the Third Provenance Challenge questions.

www.manaraa.com



(a)



(b)

Figure 5.15: The sample query results executed by *OPQL* and SQL.

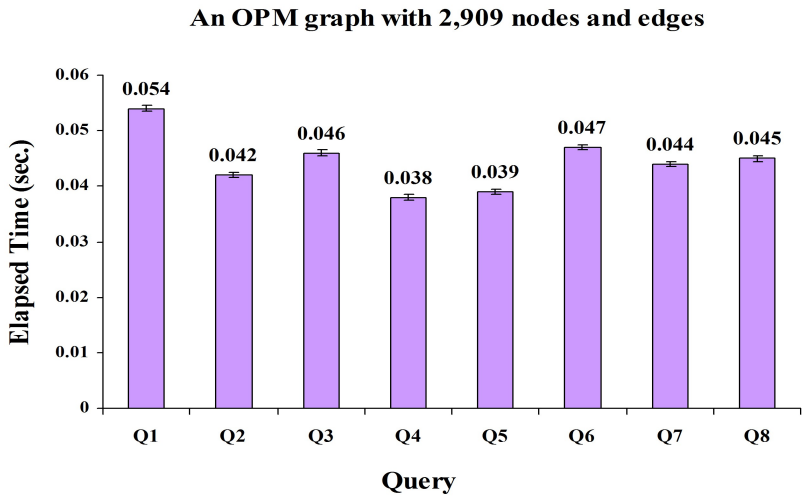
with one 2.27 GHz dual core processor and 4 GB main memory, running the Windows 7 operating system. In all the experiments, we show the results as the average of 20 trials.

5.3.1 Provenance Query Performance Experiments

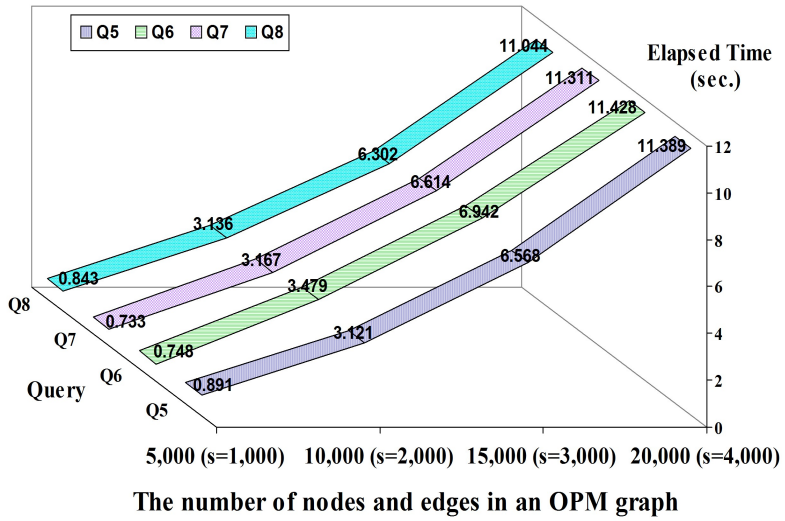
To evaluate the querying performance of *OPQL*, we used eight provenance queries depicted in Table I. These queries were executed on the dataset (UCDGC: UC Davis Genome Center), which represents an OPM graph in which the total number of nodes and edges is 2,909. The query performance experiments are reported in Figure 5.16(a). Overall, the query evaluation for *OPQL* showed to be very efficient, returning results within 0.06 seconds for all the queries (Q1-Q8).

Moreover, to explore the scalability of queries Q5, Q6, Q7, and Q8 that required the more expensive computation of transitive relationships in the OPM graph, we used four OPM-compliant

www.manaraa.com



(a) The query performance on the UCDGC dataset.



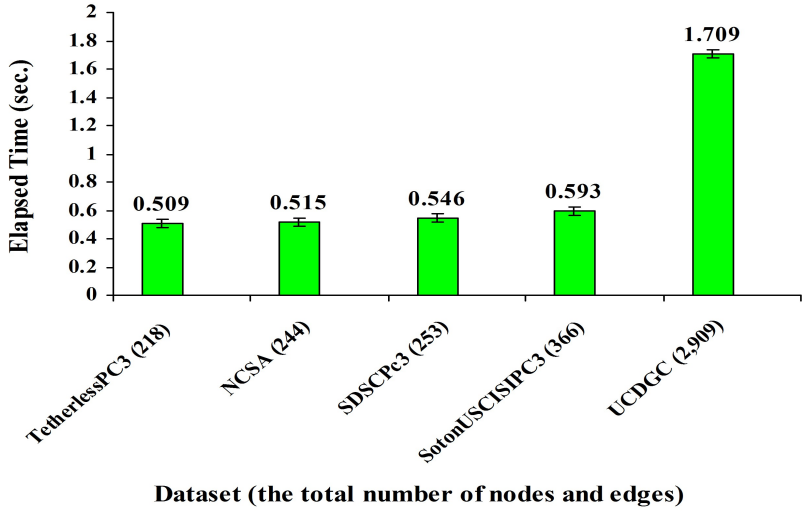
(b) The query performance on the OPM graphs with varying complexity.

Figure 5.16: *OPQL* query performance over various datasets.

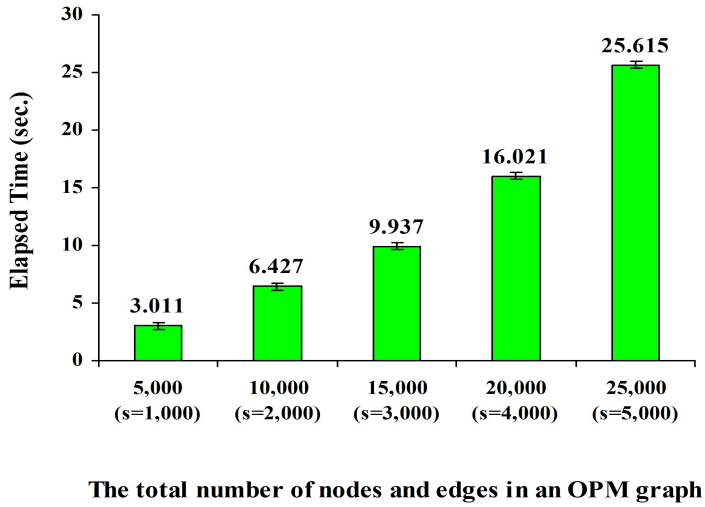
datasets generated via the simulation over five synthetic workflows, which are a sequential type of workflows (i.e., a workflow step is connected to only one workflow step) in which the total number of steps (s) is 1,000, 2,000, 3,000, and 4,000, respectively. Note that the more the number of steps of the workflow, the more expensive the computation of transitive relationships in the OPM graph. *OPQL* queries (i.e., Q5, Q6, Q7, and Q8) were evaluated on these larger datasets. The response times for these queries are reported in Figure 5.16(b). Overall, these queries showed satisfactory

www.manaraa.com

performance, returning results within around 12 seconds for the provenance dataset with 20,000 nodes and edges.



(a) The provenance visualization performance across different datasets.



(b) The provenance visualization performance on the OPM graphs with varying complexity.

Figure 5.17: OPMPROV provenance visualization performance over various datasets.

www.manaraa.com

5.3.2 Provenance Visualization Performance Experiments

To perform provenance visualization experiments, we selected five OPM-compliant provenance datasets (which represent the OPM graphs) generated by different participants of the Third Provenance Challenge [15], and then we inserted these datasets into OPMPROV. The provenance visualization performance for these datasets is reported in Figure 5.17(a), where the datasets are shown in the ascending order of the total number of nodes and edges. The results for provenance visualization showed to visualize all the datasets in less than 2 seconds.

To explore the provenance visualization performance and scalability on larger datasets, we used five OPM-compliant provenance datasets which represent the OPM graphs with varying complexity in which the total number of nodes and edges is 5,000, 10,000, 15,000, 20,000, and 25,000, respectively. The results for these datasets are reported in Figure 5.17(b). Overall, the provenance visualization performance over the larger datasets showed satisfactory performance, returning results within around 26 seconds for the dataset with 25,000 nodes and edges.

5.4 Summary

In this chapter, we designed the *OPQL* query language, including six types of graph patterns, an OPM-based graph algebra, and *OPQL* syntax and semantics, that efficiently supports provenance queries. We then implemented *OPQL* as a Web service via our OPMPROV system; therefore, users can invoke the Web service to execute *OPQL* queries in a user-friendly GUI, OPMPROVIS. Finally, we conducted experiments to evaluate the performance and feasibility of OPMPROV on *OPQL* provenance querying, and the experimental results showed satisfactory performance. To our best knowledge, *OPQL* is the first OPM-level query language and OPM-compliant provenance querying service for scientific workflows.

www.manaraa.com

CHAPTER 6

DESIGN AND IMPLEMENTATION OF OPMPROV

In this chapter, we first discuss an overall architecture of the OPMPROV system, which is a relational database-based provenance system that supports both prospective and retrospective provenance. We then discuss how our OPMPROV system, including the *OPQL* query language can be implemented.

6.1 Architecture of OPMPROV

OPMPROV is a relational database-based scientific workflow provenance system that efficiently stores, reasons, and queries prospective provenance and retrospective provenance, which is OPM-compliant provenance data (XML data that conforms to the OPM XML schema). As recognized, the design of OPMPROV has been motivated by the OPM model [18], which is a standard provenance model to facilitate and promote provenance interoperability among heterogeneous systems. OPMPROV uses the OPM model as a conceptual data model to design a native OPM provenance store, and therefore an input or output of OPMPROV is OPM-compliant provenance data. OPMPROV is compliant with the OPM model (v1.1) [18]; therefore, without any transformation between the OPM model and our provenance model, provenance data represented in XML documents, which conform to the XML schema specification for the OPM model, can be inserted into OPMPROV using a data mapping procedure that shreds the XML documents into relational tuples and stores them in the corresponding relational tables in OPMPROV. Moreover, OPMPROV can sufficiently support provenance reasoning (i.e., by completion rules and multi-step inferences) defined in the OPM model using recursive views and SQL queries alone without any additional reasoning engine.

www.manaraa.com

To give a better understanding of the design of OPMPROV, we describe an architecture of the OPMPROV system as shown in Figure 6.1, where the OPMPROV system interacts with the VIEW system. Figure 6.1(a) depicts an architecture of OPMPROV. The OPMPROV system plays a role as *Provenance Manager* of the VIEW system [5], a scientific workflow management system, which consists of six major functional subsystems, including *Workbench*, *Workflow Engine*, *Worklfow Monitor*, *Data Product Manager*, *Task Manager*, and *Provenance Manager*, as shown in Figure 6.1(b). OPMPROV has a three-layer architecture. The provenance presentation layer

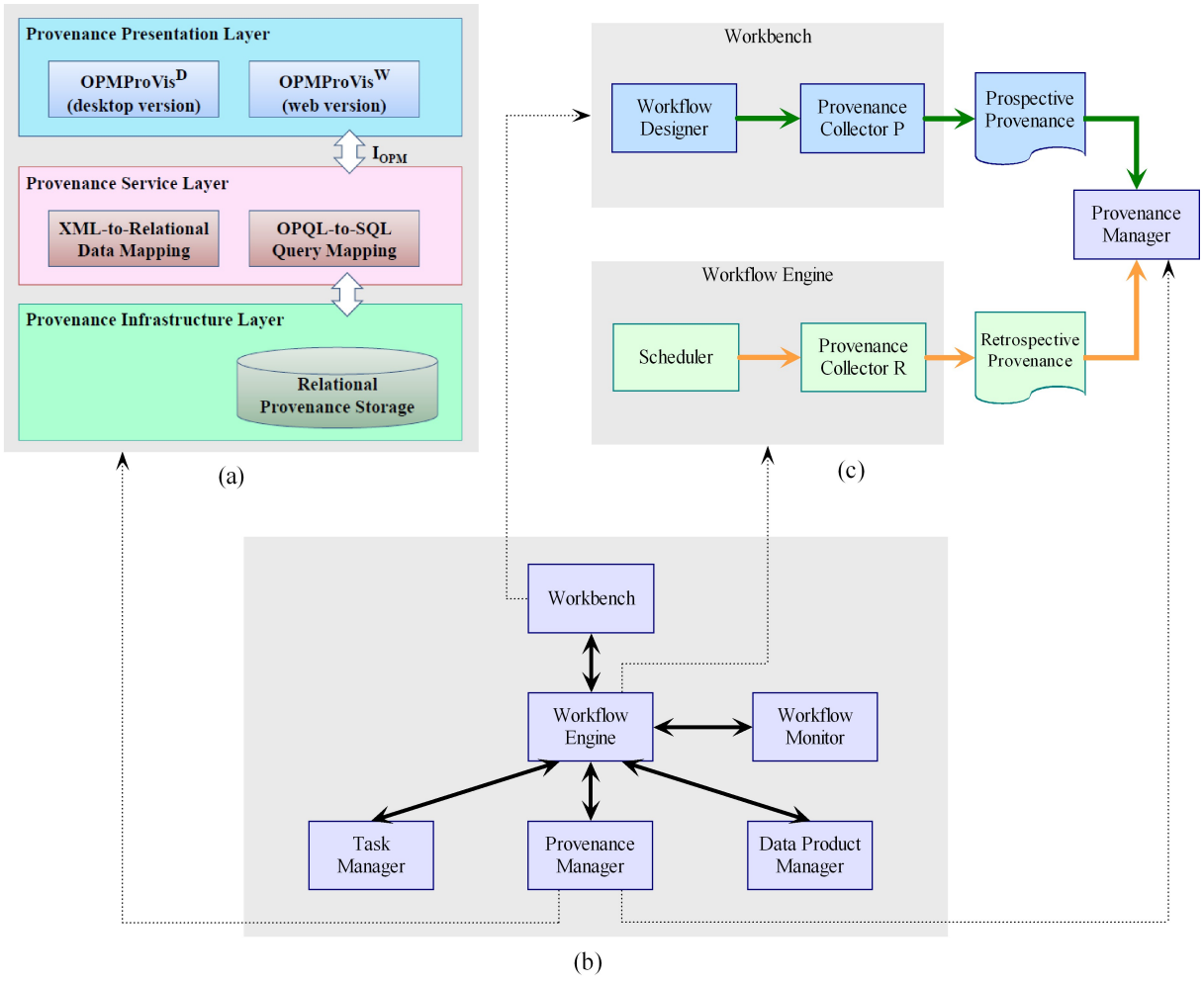


Figure 6.1: An overview of the OPMPROV system.

www.manaraa.com

provides users with the functionality of data insertion, provenance querying, and provenance visualization via user-friendly GUIs. OPMPROV supports OPMPROVIS^{D}(desktop version) and OPMPROVIS^{W}(web version) as user-friendly GUIs, which enable users to invoke Web services via interface I_{OPM} that is defined and described by WSDL (for example, the WSDL definition of the *OPQL* Web service can be found in Appendix C). The provenance service layer provides users with OPM-compliant provenance services. Currently, the OPMPROV system provides two Web services that employ two mappings; one is to insert OPM-compliant provenance data into the OPMPROV store using XML-to-Relational data mapping that maps OPM-compliant XML documents to relational tuples and the other is to execute *OPQL* queries from the OPMPROV store using OPQL-to-SQL query mapping that translates OPQL queries into SQL queries. These mappings interconnect the provenance service layer and the provenance infrastructure layer, where the latter is represented by a relational database management system that plays a role as an efficient relational provenance storage backend. In addition, as shown in Figure 6.1(c), via our proposed provenance collection framework that interacts with the VIEW system, OPMPROV can store and manage both prospective and retrospective provenance. Our OPMPROV features the native support of the OPM model.

6.2 Implementation of OPMPROV

In this section, we discuss how our OPMPROV can be implemented to query and manage prospective provenance and retrospective provenance, which is OPM-complaint provenance. Basically, we use relational database technologies to efficiently store and query provenance data. The provenance store for OPMPROV is implemented using RDBMS DB2 (v9.7.0.441). As discussed in Chapter 3, both the relational schema for prospective provenance which includes nine relations and the relational schema for retrospective provenance which includes total 33 relations are created in DB2, respectively. The details on the database schema of OPMPROV are described in Appendix A and B. To support the insertion of OPM-compliant provenance data (i.e., XML data that conforms to the OPM XML schema), we implement the *OPMXMLInsert* algorithm discussed in Chapter 4 as a

www.manaraa.com

Web service; therefore, users can store OPM-compliant provenance data into our OPMPROV store without any transformation procedure via a user-friendly GUI.

Algorithm 6 OPMGraphConstruct

01: **Input**: relational databases storing OPM-compliant provenance data
02: **Output**: OPM graph with account views G
03: **Begin**

// **Step 1: Get OPM graph entities**
04: **Let** R_a, R_p, R_{ag}, R_u, R_g, R_d, R_c, R_t, and R_{ov} be empty sets
05: **Let** T_a be a temporary list set
06: $T_a \leftarrow$ *Select (a, acc) From ArtifactHasAccount*
07: **For** each tuple t *(t.a, t.acc)* $\in T_a$ **do**
08: **If** not *overlapped (t.a)* **then**
09: $R_a \leftarrow$ *(t.a, t.acc)*
10: **Else**
11: **If** not *exists* (R_{ov}(*t.a*)) then
12: $R_{ov} \leftarrow$ *(t.a, "overlap")*
13: **End If**
14: **End If**
15: **End For**
16: $R_p \leftarrow$ *Select (p, acc) From ProcessHasAccount*
17: $R_{ag} \leftarrow$ *Select (ag, acc) From AgentHasAccount*
18: $R_u \leftarrow$ *Select (p, a, acc) From UsedHasAccount*
19: $R_g \leftarrow$ *Select (a, p, acc) From WasGeneratedByHasAccount*
20: $R_d \leftarrow$ *Select* (a_1, a_2, *acc) From WasDerivedFromHasAccount*
21: $R_c \leftarrow$ *Select (p, ag, acc) From WasControlledByHasAccount*
22: $R_t \leftarrow$ *Select* (p_1, p_2, *acc) From WasTriggeredBy*

// **Step 2: Create OPM graph vertices**
23: **Let** V_a, V_p, V_{ag}, and V_{ov} be empty sets
24: **For** each tuple t *(t.a, t.acc)* $\in R_a$ **do**
25: $V_a[] \leftarrow t.a$; $V_a[]$.tag$\leftarrow t.acc$
26: **End For**
27: **For** each tuple t *(t.a, t.acc)* $\in R_{ov}$ **do**
// get the parent and child acconts over an overlapping node
28: $V_{ov}[] \leftarrow t.a$; $V_{ov}[]$.tag$\leftarrow$ *(pacc, cacc)*
29: **End For**
30: **For** each tuple t *(t.p, t.acc)* $\in R_p$ **do**
31: $V_p[] \leftarrow t.p$; $V_p[]$.tag$\leftarrow t.acc$
32: **End For**
33: **For** each tuple t *(t.ag, t.acc)* $\in R_{ag}$ **do**
34: $V_{ag}[] \leftarrow t.ag$; $V_{ag}[]$.tag$\leftarrow t.acc$
35: **End For**

// **Step 3: Create OPM graph edges**
36: **Let** E_u, E_g, E_d, E_c, and E_t be empty sets
37: **Let** φ*(v, acc)* be a function to find index of vertex v with account *acc*, such that (*v, acc*) $\in R_a$, R_p, R_{ag}, R_{ov}
38: **For** each tuple t *(t.p, t.a, t.acc)* $\in R_u$ **do**
39: **If** not *overlapped (t.a)* **then**
40: $s = \varphi$*(t.p, t.acc)*
41: $d = \varphi$*(t.a, t.acc)*
42: $E_u[] \leftarrow (V_p[s], V_a[d])$; $E_u[]$.tag$\leftarrow t.acc$
43: **Else**
44: $s = \varphi$*(t.p, t.acc)*
45: $d = \varphi$*(t.a, "overlap")*
46: $E_u[] \leftarrow (V_p[s], V_{ov}[d])$; $E_u[]$.tag$\leftarrow t.acc$
47: **End If**
48: **End For**
49: **For** each tuple t *(t.a, t.p, t.acc)* $\in R_g$ **do**
50: **If** not *overlapped (t.a)* **then**
51: $s = \varphi$*(t.a, t.acc)*
52: $d = \varphi$*(t.p, t.acc)*
53: $E_g[] \leftarrow (V_a[s], V_p[d])$; $E_g[]$.tag$\leftarrow t.acc$
54: **Else**
55: $s = \varphi$*(t.a, "overlap")*
56: $d = \varphi$*(t.p, t.acc)*
57: $E_g[] \leftarrow (V_{ov}[s], V_p[d])$; $E_g[]$.tag$\leftarrow t.acc$
58: **End If**
59: **End For**
60: //create datasets E_d, E_c, and E_t in a similar fashion

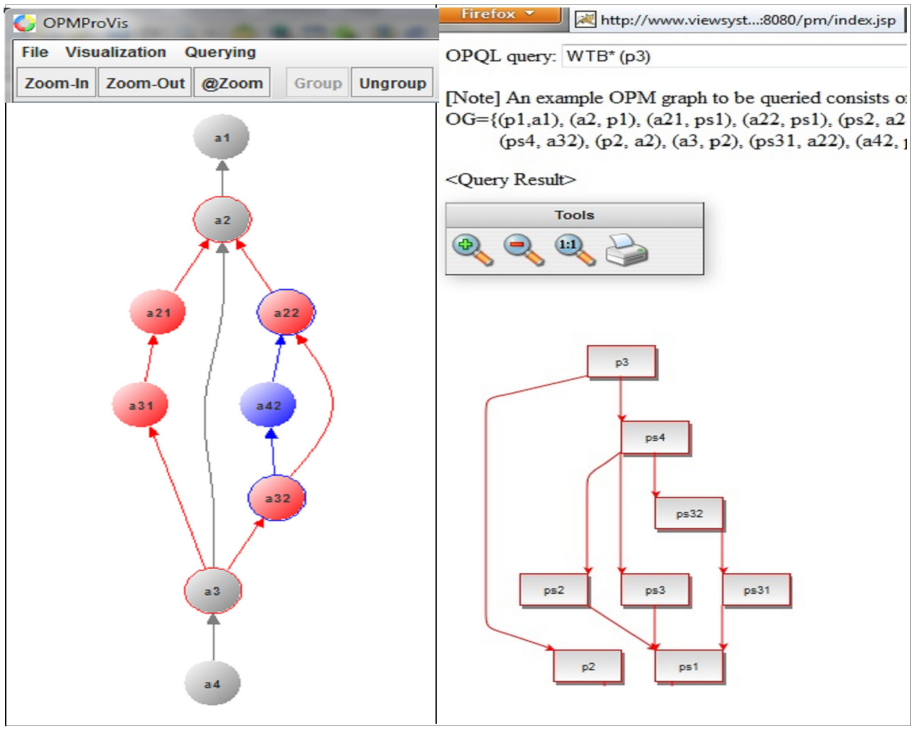
// **Step 4: Create an OPM graph with account views**
61: Get a dataset R *(account, level)* that contains the order of hierarchical account levels over overlapping accounts, where *level* is an integer
62: Select an account level l from a user interface, where $l \in R.level$
63: G $\leftarrow$ visualizeOPMgraph (l) // default is the maximum account level
// a function to visualize different levels of account views
64: **function** visualizeOPMgraph (int l)
65: **Let** account view V be an empty set
66: **For** int i=0; $i <= l$; i++ **do**
// insert the set of vertices and egdes into account view V
67: $V \leftarrow \{n \in V_a, V_p, V_{ag}, E_u, E_g, E_d, E_c, E_t \mid n$ has the same account level as $i\}$
68: $V \leftarrow \{n \in V_{ov} \mid$ the parent account level of n is the same as $i\}$
69: **End For**
70: **Return** V
71: **End**

Moreover, to efficiently support query processing of OPM-compliant provenance, we implement the *OPQL* query language as a Web service using Java and Axis2 on top of the OPMPROV system. The *OPQL* Web service takes an *OPQL* query as input, translates an *OPQL* query to an equivalent SQL query and executes the SQL query translated in OPMPROV, and returns an OPM graph as output. To invoke these Web services, we implement two kinds of user-friendly GUIs (i.e., provenance browsers) that allow users to store OPM-compliant provenance data and

www.manaraa.com

execute *OPQL* queries: one is OPMPROVIS^D (desktop version) implemented by Java and JGraph and the other is OPMPROVIS^W (web version) implemented by JSP and mxGraph. Since, in the OPMPROV system, efficient provenance visualization, either as part of visualizing a whole OPM graph or as part of visualizing the query result is important to reduce response time for provenance querying and visualization, we employ an efficient algorithm, called *OPMGraphConstruct*, to construct and visualize an OPM graph.

Algorithm 6 shows an efficient algorithm that takes relevant OPM entities from our OPMPROV store and creates an OPM graph with different levels of account views (an account view means a view of an OPM graph depending on one account). The algorithm has four steps. As the first step, *OPMGraphConstruct* takes as an input OPM-compliant relational databases, retrieves relevant OPM graph entities from the corresponding tables, and creates datasets R_a, R_p, R_{ag}, R_u,



(a) (b)

Figure 6.2: Visualizing OPM graphs in OPMPROVIS^D and OPMPROVIS^W.

www.manaraa.com

R_g, R_d, R_c, and R_t that correspond to three OPM nodes and five OPM edges, respectively. In particular, to handle overlapping accounts over an artifact, separate dataset R_{ov} is created via a condition to check if an artifact has two different accounts. The second step is responsible for creating vertices to be displayed in an OPM graph. *OPMGraphConstruct* creates vertex datasets V_a, V_p, V_{ag}, in which each vertex holds one account information. In particular, in the V_{ov} dataset that represents vertices for overlapping artifacts, each vertex holds both parent and child account information. The third step is responsible for creating edges to be displayed in an OPM graph. At lines 38-48, *OPMGraphConstruct* finds indexes of two vertices (i.e., a process vertex as source and an artifact vertex as destination) for an *Used* edge and creates an *Used* edge via the found indexes. Each edge created also holds the corresponding account information. Through iteration of creation of an *Used* edge, dataset E_u for the *Used* edges is created. Similarly, lines 49-59 demonstrates creation of the *WasGeneratedBy* edges, where indexes of two vertices (i.e., an artifact vertex as source and a process vertex as destination) for a *WasGeneratedBy* edge are found and finally dataset E_g is created. In a similar fashion, datasets E_d, E_c, and E_t can be created for edges *WasDerivedFrom*, *WasControlledBy*, and *WasTriggeredBy*, respectively. Finally, the fourth step is responsible for creating an OPM graph with different levels of account views based on the created vertices and edges. *OPMGraphConstruct* first computes the order of hierarchical account levels over overlapping accounts via recursive queries using additional relation *OverlapAccount* and creates dataset *R (account, level)* that contains the account level corresponding one account, where one account level has an integer value. Using the account level (default is the maximum value of account levels) selected by a user interface, an OPM graph with different account views is created and visualized via a function that finds and collects vertices and edges whose account level is the same as the account level until the integration condition of the account level taken as an argument meets.

This algorithm is implemented in our user-friendly GUIs (OPMProVis^D and OPMProVis^W) to visualizes not only a whole OPM graph but also the result of an *OPQL* query. Figure 6.2(a) shows the output of *OPQL* query *WDF**(a_4) in OPMProVis^D and Figure 6.2(b) shows the output of *OPQL* query *WTB**(p_3) in OPMProVis^W.

www.manaraa.com

CHAPTER 7
CONCLUSIONS AND FUTURE WORK

We conclude this dissertation by summarizing our results, discussing our contributions and presenting future work.

7.1 Summary

In this dissertation, we first proposed a provenance collection framework to capture both prospective and retrospective provenance. We then proposed a relational database-based provenance system that stores, reasons, and queries prospective and retrospective provenance. Then, we proposed OPQL, an OPM-level provenance query language, that is directly defined over the Open Provenance Model (OPM), which is a standard provenance model. Finally, we presented the design and implementation of the OPMPROV system.

7.2 Contributions

Our contributions are elaborated in detail as follows:

- First, to propose a provenance collection mechanism that captures both prospective and retrospective provenance in scientific workflow environments, (i) we designed a provenance model that models both prospective and retrospective provenance as an extension to the Open Provenance Model (OPM), which only models retrospective provenance and (ii) we proposed a provenance collection framework to collect both prospective and retrospective provenance according to our model. While most existing systems use an internal proprietary provenance model and develop an import/export facility to convert between the proprietary model and OPM, our provenance collection framework features the native support of the OPM model.

www.manaraa.com

- Second, to propose a relational database-based provenance system, called OPMPROV that stores, reasons, and queries prospective and retrospective provenance, which is compliant with the OPM model, (i) we designed a relational database schema for the storage of provenance and (ii) we showed that provenance reasoning defined in the OPM model (v1.1) can be sufficiently supported by OPMPROV using recursive views and SQL queries alone without any additional reasoning engine. Experiments are conducted to evaluate the performance of OPMPROV in data insertion and provenance querying. A case study is performed, demonstrating that OPMPROV can answer all except one query out of the 16 queries defined in the Third Provenance Challenge.

- Third, to design *OPQL*, an OPM-level provenance query language, that is directly defined over the Open Provenance Model (OPM), (i) we designed *OPQL*, including six types of graph patterns, an OPM-based graph algebra, and *OPQL* syntax and semantics, that efficiently supports provenance queries and (ii) we implemented *OPQL* using a Web service via our OPMPROV system; therefore, users can invoke the Web service to execute *OPQL* queries in a provenance browser, called OPMPROVIS. The result of *OPQL* queries is displayed as an OPM graph in OPMPROVIS. An experimental study is conducted to evaluate the feasibility and performance of OPMPROV on *OPQL* provenance querying. To our best knowledge, *OPQL* is the first OPM-level query language and OPM-compliant provenance querying service for scientific workflows.

7.3 Future Work

This dissertation has addressed several issues associated with querying and managing OPM-compliant provenance data in scientific workflows. These issues are expected to result in some additional problems and solutions from the scientific workflow provenance community. This section describes possible future work as follows:

www.manaraa.com

- We showed that provenance reasoning defined in the OPM model can be sufficiently supported by OPMPROV using recursive views and SQL queries alone without additional reasoning engine. In reality, it is expensive to use recursive views for provenance reasoning in case of large amounts of provenance data. Thus, presenting a mechanism to improve the performance of provenance reasoning and querying in OPMPROV is considered as a potential future work.

- We desinged *OPQL*, including six types of graph patterns, an OPM-based graph algebra, and *OPQL* syntax and semantics, that efficiently supports provenance queries. To enhance the accessibility of *OPQL*, we implemented *OPQL* as an OPM-compliant provenance service for scientific workflows using a Web service. As a future work, evaluating *OPQL* from other points of view, including expressiveness, completeness, and usability is considered.

- We performed our experiments to evaluate the performance of OPMPROV in data insertion and provenance querying. To efficiently support large amounts of provenance data in scientific workflow provenance management, using cloud computing technologies provides many benefits in term of reliability, usability, scalability, performance, and maintenance. Thus, exploring a cloud-based provenance store is considered as a potential future work.

www.manaraa.com

APPENDIX A

Relational Schema for Prospective Provenance in OPMProv

```
CREATE TABLE Workflow (
    WorkflowId VARCHAR(50) NOT NULL,
    Description VARCHAR(255),
    PRIMARY KEY (WorkflowId)
);

CREATE TABLE Task (
    TaskId VARCHAR(50) NOT NULL,
    TaskName VARCHAR(255),
    TaskType VARCHAR(50),
    PRIMARY KEY (TaskId)
);

CREATE TABLE Performer (
    PFId VARCHAR(50) NOT NULL,
    PFName VARCHAR(255),
    PRIMARY KEY (PFId)
);
```

www.manaraa.com

```
CREATE TABLE Port (
    PortId VARCHAR(50) NOT NULL,
    TaskId VARCHAR(50),
    PortName VARCHAR(255),
    PortType VARCHAR(50),
    PRIMARY KEY (PortId),
    CONSTRAINT fk_Task
    FOREIGN KEY (TaskId) REFERENCES Task (TaskId) ON DELETE CASCADE
);

CREATE TABLE Contains (
    ParentTaskId VARCHAR(50) NOT NULL,
    ChildTaskId VARCHAR(50) NOT NULL,
    PRIMARY KEY (ParentTaskId, ChildTaskId),
    CONSTRAINT fk_Task
    FOREIGN KEY (ParentTaskId) REFERENCES Task (TaskId) ON DELETE CASCADE,
    CONSTRAINT fk_Task
    FOREIGN KEY (ChildTaskId) REFERENCES Task (TaskId) ON DELETE CASCADE
);
```

www.manaraa.com

```
CREATE TABLE Performs (
    PFId VARCHAR(50) NOT NULL,
    TaskId VARCHAR(50) NOT NULL,
    PRIMARY KEY (PFId, TaskId),
    CONSTRAINT fk_Performer
    FOREIGN KEY (PFId) REFERENCES Performer (PFId) ON DELETE CASCADE,
    CONSTRAINT fk_Task
    FOREIGN KEY (TaskId) REFERENCES Task (TaskId) ON DELETE CASCADE
);

CREATE TABLE IsConnectedTo (
    SourcePortId VARCHAR(50) NOT NULL,
    DestinationPortId VARCHAR(50) NOT NULL,
    PRIMARY KEY (SourcePortId, DestinationPortId),
    CONSTRAINT fk_Port
    FOREIGN KEY (SourcePortId) REFERENCES Port (PortId) ON DELETE CASCADE,
    CONSTRAINT fk_Port
    FOREIGN KEY (DestinationPortId) REFERENCES Port (PortId) ON DELETE CASCADE
);
```

www.manaraa.com

```
CREATE TABLE TaskPartOfWorkflow (
    TaskId VARCHAR(50) NOT NULL,
    WorkflowId VARCHAR(50) NOT NULL,
    PRIMARY KEY (TaskId, WorkflowId),
    CONSTRAINT fk_Task
    FOREIGN KEY (TaskId) REFERENCES Task (TaskId) ON DELETE CASCADE,
    CONSTRAINT fk_Workflow
    FOREIGN KEY (WorkflowId) REFERENCES Workflow (WorkflowId) ON DELETE CASCADE
);

CREATE TABLE PerformerPartOfWorkflow (
    PFId VARCHAR(50) NOT NULL,
    WorkflowId VARCHAR(50) NOT NULL,
    PRIMARY KEY (PFId, WorkflowId),
    CONSTRAINT fk_Performer
    FOREIGN KEY (PFId) REFERENCES Performer (PFId) ON DELETE CASCADE,
    CONSTRAINT fk_Workflow
    FOREIGN KEY (WorkflowId) REFERENCES Workflow (WorkflowId) ON DELETE CASCADE
);
```

www.manaraa.com

APPENDIX B

Relational Schema for Retrospective Provenance in OPMPROV

```
CREATE TABLE OPMGraph (
    OPMGraphId VARCHAR(50) NOT NULL,
    WorkflowId VARCHAR(50) ,
    PRIMARY KEY (OPMGraphId),
    CONSTRAINT fk_Workflow
    FOREIGN KEY (WorkflowId) REFERENCES Workflow (WorkflowId)
    ON DELETE CASCADE
);

CREATE TABLE OPMGraphAnnotation (
    OPMGraphId VARCHAR(50) NOT NULL,
    Property VARCHAR(255) NOT NULL,
    Value VARCHAR(255) NOT NULL,
    PRIMARY KEY (OPMGraphId, Property, Value),
    CONSTRAINT fk_OPMGraph
    FOREIGN KEY (OPMGraphId) REFERENCES OPMGraph (OPMGraphId)
    ON DELETE CASCADE
);
```

www.manaraa.com

```
CREATE TABLE Artifact (
    OPMGraphId VARCHAR(50) NOT NULL,
    ArtifactId VARCHAR(50) NOT NULL,
    Value VARCHAR(255),
    PRIMARY KEY (OPMGraphId, ArtifactId),
    CONSTRAINT fk_OPMGraph
    FOREIGN KEY (OPMGraphId) REFERENCES OPMGraph (OPMGraphId)
    ON DELETE CASCADE
);

CREATE TABLE ArtifactHasAccount(
    OPMGraphId VARCHAR(50) NOT NULL,
    ArtifactId VARCHAR(50) NOT NULL,
    Account VARCHAR(50) NOT NULL,
    PRIMARY KEY (OPMGraphId, ArtifactId, Account),
    CONSTRAINT fk_Artifact1
    FOREIGN KEY (OPMGraphId, ArtifactId) REFERENCES Artifact (OPMGraphId, ArtifactId)
    ON DELETE CASCADE
);
```

www.manaraa.com

```
CREATE TABLE Process (
    OPMGraphId VARCHAR(50) NOT NULL,
    ProcessId VARCHAR(50) NOT NULL,
    Value VARCHAR(255),
    TaskId VARCHAR(50),
    PRIMARY KEY (OPMGraphId, ProcessId),
    CONSTRAINT fk_OPMGraph
    FOREIGN KEY (OPMGraphId) REFERENCES OPMGraph (OPMGraphId)
    ON DELETE CASCADE,
    CONSTRAINT fk_Task
    FOREIGN KEY (TaskId) REFERENCES Task (TaskId)
    ON DELETE CASCADE
);

CREATE TABLE ProcessHasAccount(
    OPMGraphId VARCHAR(50) NOT NULL,
    ProcessId VARCHAR(50) NOT NULL,
    Account VARCHAR(50) NOT NULL,
    PRIMARY KEY (OPMGraphId, ProcessId, Account),
    CONSTRAINT fk_Process
    FOREIGN KEY (OPMGraphId, ProcessId) REFERENCES Process (OPMGraphId, ProcessId)
    ON DELETE CASCADE
);
```

www.manaraa.com

```
CREATE TABLE Agent (
    OPMGraphId VARCHAR(50) NOT NULL,
    AgentId VARCHAR(50) NOT NULL,
    Value VARCHAR(255),
    PFId VARCHAR(50),
    PRIMARY KEY (OPMGraphId, AgentId),
    CONSTRAINT fk_OPMGraph
    FOREIGN KEY (OPMGraphId) REFERENCES OPMGraph (OPMGraphId)
    ON DELETE CASCADE,
    CONSTRAINT fk_Performer
    FOREIGN KEY (PFId) REFERENCES Performer (PFId)
    ON DELETE CASCADE
);

CREATE TABLE AgentHasAccount (
    OPMGraphId VARCHAR(50) NOT NULL,
    AgentId VARCHAR(50) NOT NULL,
    Account VARCHAR(50) NOT NULL,
    PRIMARY KEY (OPMGraphId, AgentId, Account),
    CONSTRAINT fk_Agent
    FOREIGN KEY (OPMGraphId, AgentId) REFERENCES Agent (OPMGraphId, AgentId)
    ON DELETE CASCADE
);
```

www.manaraa.com

```
CREATE TABLE WasInputTo (
    OPMGraphId VARCHAR(50) NOT NULL,
    ArtifactId VARCHAR(50) NOT NULL,
    PortId VARCHAR(50) NOT NULL,
    PRIMARY KEY (OPMGraphId, ArtifactId, PortId),
    CONSTRAINT fk_Artifact
    FOREIGN KEY (OPMGraphId, ArtifactId) REFERENCES Artifact (OPMGraphId, ArtifactId)
    ON DELETE CASCADE,
    CONSTRAINT fk_Port
    FOREIGN KEY (PortId) REFERENCES Port (PortId) ON DELETE CASCADE
);

CREATE TABLE WasOutputBy (
    OPMGraphId VARCHAR(50) NOT NULL,
    ArtifactId VARCHAR(50) NOT NULL,
    PortId VARCHAR(50),
    PRIMARY KEY (OPMGraphId, ArtifactId),
    CONSTRAINT fk_Artifact
    FOREIGN KEY (OPMGraphId, ArtifactId) REFERENCES Artifact (OPMGraphId, ArtifactId)
    ON DELETE CASCADE,
    CONSTRAINT fk_Port
    FOREIGN KEY (PortId) REFERENCES Port (PortId)
    ON DELETE CASCADE
);
```

www.manaraa.com

```
CREATE TABLE Used (
    OPMGraphId VARCHAR(50) NOT NULL,
    ProcessId VARCHAR(50) NOT NULL,
    Role VARCHAR(255) NOT NULL,
    ArtifactId VARCHAR(50) NOT NULL,
    OTimeLower VARCHAR(50),
    OTimeUpper VARCHAR(50),
    PRIMARY KEY (OPMGraphId, ProcessId, Role, ArtifactId),
    CONSTRAINT fk_Process
    FOREIGN KEY (OPMGraphId, ProcessId) REFERENCES Process (OPMGraphId, ProcessId)
    ON DELETE CASCADE,
    CONSTRAINT fk_Artifact
    FOREIGN KEY (OPMGraphId, ArtifactId) REFERENCES Artifact (OPMGraphId, ArtifactId)
    ON DELETE CASCADE
);

CREATE TABLE UsedHasAccount (
    OPMGraphId VARCHAR(50) NOT NULL,
    ProcessId VARCHAR(50) NOT NULL,
    Role VARCHAR(255) NOT NULL,
    ArtifactId VARCHAR(50) NOT NULL,
    Account VARCHAR(50) NOT NULL,
    PRIMARY KEY (OPMGraphId, ProcessId, Role, ArtifactId, Account),
    CONSTRAINT fk_Used
    FOREIGN KEY (OPMGraphId, ProcessId, Role, ArtifactId)
    REFERENCES Used (OPMGraphId, ProcessId, Role, ArtifactId) ON DELETE CASCADE
);
```

www.manaraa.com

```
CREATE TABLE WasGeneratedBy (
    OPMGraphId VARCHAR(50) NOT NULL,
    ArtifactId VARCHAR(50) NOT NULL,
    Role VARCHAR(255) NOT NULL,
    ProcessId VARCHAR(50) NOT NULL,
    OTimeLower VARCHAR(50),
    OTimeUpper VARCHAR(50),
    PRIMARY KEY (OPMGraphId, ArtifactId, Role, ProcessId),
    CONSTRAINT fk_Artifact
    FOREIGN KEY (OPMGraphId, ArtifactId) REFERENCES Artifact (OPMGraphId, ArtifactId)
    ON DELETE CASCADE,
    CONSTRAINT fk_Process
    FOREIGN KEY (OPMGraphId, ProcessId) REFERENCES Process (OPMGraphId, ProcessId)
    ON DELETE CASCADE
);

CREATE TABLE WasGeneratedByHasAccount (
    OPMGraphId VARCHAR(50) NOT NULL,
    ArtifactId VARCHAR(50) NOT NULL,
    Role VARCHAR(255) NOT NULL,
    ProcessId VARCHAR(50) NOT NULL,
    Account VARCHAR(50) NOT NULL,
    PRIMARY KEY (OPMGraphId, ArtifactId, Role, ProcessId, Account),
    CONSTRAINT fk_WasGeneratedBy
    FOREIGN KEY (OPMGraphId, ArtifactId, Role, ProcessId)
    REFERENCES WasGeneratedBy (OPMGraphId, ArtifactId, Role, ProcessId)
    ON DELETE CASCADE
);
```

www.manaraa.com

```
CREATE TABLE WasControlledBy (
    OPMGraphId VARCHAR(50) NOT NULL,
    ProcessId VARCHAR(50) NOT NULL,
    Role VARCHAR(255) NOT NULL,
    AgentId VARCHAR(50) NOT NULL,
    OTimeStartLower VARCHAR(50),
    OTimeStartUpper VARCHAR(50),
    OTimeEndLower VARCHAR(50),
    OTimeEndUpper VARCHAR(50),
    PRIMARY KEY (OPMGraphId, ProcessId, Role, AgentId),
    CONSTRAINT fk_Process
    FOREIGN KEY (OPMGraphId, ProcessId) REFERENCES Process (OPMGraphId, ProcessId)
    ON DELETE CASCADE,
    CONSTRAINT fk_Agent
    FOREIGN KEY (OPMGraphId, AgentId) REFERENCES Agent (OPMGraphId, AgentId)
    ON DELETE CASCADE
);

CREATE TABLE WasControlledByHasAccount (
    OPMGraphId VARCHAR(50) NOT NULL,
    ProcessId VARCHAR(50) NOT NULL,
    Role VARCHAR(255) NOT NULL,
    AgentId VARCHAR(50) NOT NULL,
    Account VARCHAR(50) NOT NULL,
    PRIMARY KEY (OPMGraphId, ProcessId, Role, AgentId, Account),
    CONSTRAINT fk_WasControlledBy
    FOREIGN KEY (OPMGraphId, ProcessId, Role, AgentId)
    REFERENCES WasControlledBy (OPMGraphId, ProcessId, Role, AgentId)
    ON DELETE CASCADE
);
```

www.manaraa.com

```
CREATE TABLE WasDerivedFrom (
    OPMGraphId VARCHAR(50) NOT NULL,
    EffectArtifactId VARCHAR(50) NOT NULL,
    CauseArtifactId VARCHAR(50) NOT NULL,
    OTimeLower VARCHAR(50),
    OTimeUpper VARCHAR(50),
    PRIMARY KEY (OPMGraphId, EffectArtifactId, CauseArtifactId),
    CONSTRAINT fk_Artifact
    FOREIGN KEY (OPMGraphId, EffectArtifactId)
    REFERENCES Artifact (OPMGraphId, ArtifactId) ON DELETE CASCADE,
    CONSTRAINT fk_Artifact
    FOREIGN KEY (OPMGraphId, CauseArtifactId)
    REFERENCES Artifact (OPMGraphId, ArtifactId) ON DELETE CASCADE
);

CREATE TABLE WasDerivedFromHasAccount (
    OPMGraphId VARCHAR(50) NOT NULL,
    EffectArtifactId VARCHAR(50) NOT NULL,
    CauseArtifactId VARCHAR(50) NOT NULL,
    Account VARCHAR(50) NOT NULL,
    PRIMARY KEY (OPMGraphId, EffectArtifactId, CauseArtifactId, Account),
    CONSTRAINT fk_WasDerivedFrom
    FOREIGN KEY (OPMGraphId, EffectArtifactId, CauseArtifactId)
    REFERENCES WasDerivedFrom (OPMGraphId, EffectArtifactId, CauseArtifactId)
    ON DELETE CASCADE
);
```

www.manaraa.com

```
CREATE TABLE ExplicitWasTriggeredBy (
    OPMGraphId VARCHAR(50) NOT NULL,
    EffectProcessId VARCHAR(50) NOT NULL,
    CauseProcessId VARCHAR(50) NOT NULL,
    OTimeLower VARCHAR(50),
    OTimeUpper VARCHAR(50),
    PRIMARY KEY (OPMGraphId, EffectProcessId, CauseProcessId),
    CONSTRAINT fk_Process
    FOREIGN KEY (OPMGraphId, EffectProcessId)
    REFERENCES Process (OPMGraphId, ProcessId) ON DELETE CASCADE,
    CONSTRAINT fk_Artifact
    FOREIGN KEY (OPMGraphId, CauseProcessId)
    REFERENCES Process (OPMGraphId, ProcessId) ON DELETE CASCADE
);

CREATE TABLE ExplicitWasTriggeredByHasAccount (
    OPMGraphId VARCHAR(50) NOT NULL,
    EffectProcessId VARCHAR(50) NOT NULL,
    CauseProcessId VARCHAR(50) NOT NULL,
    Account VARCHAR(50) NOT NULL,
    PRIMARY KEY (OPMGraphId, EffectProcessId, CauseProcessId, Account),
    CONSTRAINT fk_ExplicitWasTriggeredBy
    FOREIGN KEY (OPMGraphId, EffectProcessId, CauseProcessId)
    REFERENCES ExplicitWasTriggeredBy (OPMGraphId, EffectProcessId, CauseProcessId)
    ON DELETE CASCADE
);
```

www.manaraa.com

```
CREATE TABLE ArtifactAnnotation (
    OPMGraphId VARCHAR(50) NOT NULL,
    ArtifactId VARCHAR(50) NOT NULL,
    Property VARCHAR(255) NOT NULL,
    Value VARCHAR(255) NOT NULL,
    PRIMARY KEY (OPMGraphId, ArtifactId, Property, Value),
    CONSTRAINT fk_Artifact
    FOREIGN KEY (OPMGraphId, ArtifactId) REFERENCES Artifact (OPMGraphId, ArtifactId)
    ON DELETE CASCADE
);

CREATE TABLE ProcessAnnotation (
    OPMGraphId VARCHAR(50) NOT NULL,
    ProcessId VARCHAR(50) NOT NULL,
    Property VARCHAR(255) NOT NULL,
    Value VARCHAR(255) NOT NULL,
    PRIMARY KEY (OPMGraphId, ProcessId, Property, Value),
    CONSTRAINT fk_Process
    FOREIGN KEY (OPMGraphId, ProcessId) REFERENCES Process (OPMGraphId, ProcessId)
    ON DELETE CASCADE
);
```

www.manaraa.com

```
CREATE TABLE AgentAnnotation (
    OPMGraphId VARCHAR(50) NOT NULL,
    AgentId VARCHAR(50) NOT NULL,
    Property VARCHAR(255) NOT NULL,
    Value VARCHAR(255) NOT NULL,
    PRIMARY KEY (OPMGraphId, AgentId, Property, Value),
    CONSTRAINT fk_Agent
    FOREIGN KEY (OPMGraphId, AgentId) REFERENCES Agent (OPMGraphId, AgentId)
    ON DELETE CASCADE
);

CREATE TABLE UsedAnnotation (
    OPMGraphId VARCHAR(50) NOT NULL,
    ProcessId VARCHAR(50) NOT NULL,
    Role VARCHAR(255) NOT NULL,
    ArtifactId VARCHAR(50) NOT NULL,
    Property VARCHAR(255) NOT NULL,
    Value VARCHAR(255) NOT NULL,
    PRIMARY KEY (OPMGraphId, ProcessId, Role, ArtifactId, Property, Value),
    CONSTRAINT fk_Used
    FOREIGN KEY (OPMGraphId, ProcessId, Role, ArtifactId)
    REFERENCES Used (OPMGraphId, ProcessId, Role, ArtifactId) ON DELETE CASCADE
);
```

www.manaraa.com

```
CREATE TABLE WasGeneratedByAnnotation (
    OPMGraphId VARCHAR(50) NOT NULL,
    ArtifactId VARCHAR(50) NOT NULL,
    Role VARCHAR(255) NOT NULL,
    ProcessId VARCHAR(50) NOT NULL,
    Property VARCHAR(255) NOT NULL,
    Value VARCHAR(255) NOT NULL,
    PRIMARY KEY (OPMGraphId, ArtifactId, Role, ProcessId, Property, Value),
    CONSTRAINT fk_WasGeneratedBy
    FOREIGN KEY (OPMGraphId, ArtifactId, Role, ProcessId)
    REFERENCES WasGeneratedBy (OPMGraphId, ArtifactId, Role, ProcessId)
    ON DELETE CASCADE
);

CREATE TABLE WasControlledByAnnotation (
    OPMGraphId VARCHAR(50) NOT NULL,
    ProcessId VARCHAR(50) NOT NULL,
    Role VARCHAR(255) NOT NULL,
    AgentId VARCHAR(50) NOT NULL,
    Property VARCHAR(255) NOT NULL,
    Value VARCHAR(255) NOT NULL,
    PRIMARY KEY (OPMGraphId, ProcessId, Role, AgentId, Property, Value),
    CONSTRAINT fk_WasControlledBy
    FOREIGN KEY (OPMGraphId, ProcessId, Role, AgentId)
    REFERENCES WasControlledBy (OPMGraphId, ProcessId, Role, AgentId)
    ON DELETE CASCADE
);
```

www.manaraa.com

```
CREATE TABLE WasDerivedFromAnnotation (
    OPMGraphId VARCHAR(50) NOT NULL,
    EffectArtifactId VARCHAR(50) NOT NULL,
    CauseArtifactId VARCHAR(50) NOT NULL,
    Property VARCHAR(255) NOT NULL,
    Value VARCHAR(255) NOT NULL,
    PRIMARY KEY (OPMGraphId, EffectArtifactId, CauseArtifactId, Property, Value),
    CONSTRAINT fk_WasDerivedFrom
    FOREIGN KEY (OPMGraphId, EffectArtifactId, CauseArtifactId)
    REFERENCES WasDerivedFrom (OPMGraphId, EffectArtifactId, CauseArtifactId)
    ON DELETE CASCADE
);

CREATE TABLE ExplicitWasTriggeredByAnnotation (
    OPMGraphId VARCHAR(50) NOT NULL,
    EffectProcessId VARCHAR(50) NOT NULL,
    CauseProcessId VARCHAR(50) NOT NULL,
    Property VARCHAR(255) NOT NULL,
    Value VARCHAR(255) NOT NULL,
    PRIMARY KEY (OPMGraphId, EffectProcessId, CauseProcessId, Property, Value),
    CONSTRAINT fk_ExplicitWasTriggeredBy
    FOREIGN KEY (OPMGraphId, EffectProcessId, CauseProcessId)
    REFERENCES ExplicitWasTriggeredBy (OPMGraphId, EffectProcessId, CauseProcessId)
    ON DELETE CASCADE
);
```

www.manaraa.com

APPENDIX C

The WSDL Definition of the OPQL Web Service

```
<?xml version="1.0" encoding="UTF-8" ?>
<wsdl:definitions xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
    xmlns:ns1="http://org.apache.axis2/xsd" xmlns:ns="http://pm.viewsystem.org"
    xmlns:wsaw="http://www.w3.org/2006/05/addressing/wsdl"
    xmlns:ax21="http://pm.viewsystem.org/xsd"
    xmlns:xs="http://www.w3.org/2001/XMLSchema"
    xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/"
    xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
    xmlns:soap12="http://schemas.xmlsoap.org/wsdl/soap12/"
    targetNamespace="http://pm.viewsystem.org">
    <wsdl:documentation>OPQL</wsdl:documentation>
    <wsdl:types>
    <xs:schema attributeFormDefault="qualified" elementFormDefault="qualified"
        targetNamespace="http://pm.viewsystem.org/xsd">
        <xs:complexType name="OPMGraphNodeFactoryAdapter">
            <xs:sequence>
                <xs:element maxOccurs="unbounded" minOccurs="0"
                    name="OPMGraphNode" nillable="true" type="ax21:IOPMGraphNode" />
            </xs:sequence>
```

www.manaraa.com

```
    </xs:complexType>
    <xs:complexType name="IOPMGraphNode">
        <xs:sequence>
            <xs:element minOccurs="0" name="account" nillable="true" type="xs:string" />
            <xs:element minOccurs="0" name="nodeId" nillable="true" type="xs:string" />
            <xs:element minOccurs="0" name="value" nillable="true" type="xs:string" />
        </xs:sequence>
    </xs:complexType>
    <xs:complexType name="OPMGraphFactoryAdapter">
        <xs:sequence>
            <xs:element maxOccurs="unbounded" minOccurs="0" name="OPMGraph"
                nillable="true" type="ax21:IOPMGraph" />
            <xs:element maxOccurs="unbounded" minOccurs="0" name="OPMGraphs"
                nillable="true" type="ax21:IOPMGraph" />
        </xs:sequence>
    </xs:complexType>
    <xs:complexType name="IOPMGraph">
        <xs:sequence>
            <xs:element minOccurs="0" name="destinationNode"
                nillable="true" type="xs:string" />
            <xs:element minOccurs="0" name="sourceNode"
                nillable="true" type="xs:string" />
        </xs:sequence>
    </xs:complexType>
</xs:schema>
```

www.manaraa.com

```
<xs:schema xmlns:ax22="http://pm.viewsystem.org/xsd" attributeFormDefault="qualified"
    elementFormDefault="qualified" targetNamespace="http://pm.viewsystem.org">
    <xs:import namespace="http://pm.viewsystem.org/xsd" />
    <xs:element name="getRelationProcessHasAccount">
        <xs:complexType>
            <xs:sequence>
                <xs:element minOccurs="0" name="opmgraphid"
                    nillable="true" type="xs:string" />
            </xs:sequence>
        </xs:complexType>
    </xs:element>
    <xs:element name="getRelationProcessHasAccountResponse">
        <xs:complexType>
            <xs:sequence>
                <xs:element minOccurs="0" name="return" nillable="true"
                    type="ax22:OPMGraphNodeFactoryAdapter" />
            </xs:sequence>
        </xs:complexType>
    </xs:element>
    <xs:element name="getRelationArtifactHasAccountWithOverlappedAccount">
        <xs:complexType>
            <xs:sequence>
                <xs:element minOccurs="0" name="opmgraphid"
                    nillable="true" type="xs:string" />
            </xs:sequence>
```

www.manaraa.com

```
    </xs:complexType>
</xs:element>
<xs:element name="getRelationArtifactHasAccountWithOverlappedAccountResponse">
    <xs:complexType>
        <xs:sequence>
            <xs:element minOccurs="0" name="return" nillable="true"
                type="ax22:OPMGraphNodeFactoryAdapter" />
        </xs:sequence>
    </xs:complexType>
</xs:element>
<xs:element name="getRelationAgentHasAccount">
    <xs:complexType>
        <xs:sequence>
            <xs:element minOccurs="0" name="opmgraphid"
                nillable="true" type="xs:string" />
        </xs:sequence>
    </xs:complexType>
</xs:element>
<xs:element name="getRelationAgentHasAccountResponse">
    <xs:complexType>
        <xs:sequence>
            <xs:element minOccurs="0" name="return" nillable="true"
                type="ax22:OPMGraphNodeFactoryAdapter" />
        </xs:sequence>
    </xs:complexType>
```

www.manaraa.com

```
    </xs:element>
    <xs:element name="executeOPQLQuery">
        <xs:complexType>
            <xs:sequence>
                <xs:element minOccurs="0" name="opmgraphid"
                    nillable="true" type="xs:string" />
                <xs:element minOccurs="0" name="opqlQuery"
                    nillable="true" type="xs:string" />
            </xs:sequence>
        </xs:complexType>
    </xs:element>
    <xs:element name="executeOPQLQueryResponse">
        <xs:complexType>
            <xs:sequence>
                <xs:element minOccurs="0" name="return" nillable="true"
                    type="ax22:OPMGraphFactoryAdapter" />
            </xs:sequence>
        </xs:complexType>
    </xs:element>
</xs:schema>
</wsdl:types>
<wsdl:message name="getRelationArtifactHasAccountWithOverlappedAccountRequest">
    <wsdl:part name="parameters"
        element="ns:getRelationArtifactHasAccountWithOverlappedAccount" />
</wsdl:message>
```

www.manaraa.com

```
<wsdl:message name="getRelationArtifactHasAccountWithOverlappedAccountResponse">
    <wsdl:part name="parameters"
        element="ns:getRelationArtifactHasAccountWithOverlappedAccountResponse" />
</wsdl:message>
<wsdl:message name="getRelationAgentHasAccountRequest">
    <wsdl:part name="parameters" element="ns:getRelationAgentHasAccount" />
</wsdl:message>
<wsdl:message name=
    "getRelationAgentHasAccountResponse">
    <wsdl:part name="parameters" element="ns:getRelationAgentHasAccountResponse" />
</wsdl:message>
<wsdl:message name="getRelationProcessHasAccountRequest">
    <wsdl:part name="parameters" element="ns:getRelationProcessHasAccount" />
</wsdl:message>
<wsdl:message name="getRelationProcessHasAccountResponse">
    <wsdl:part name="parameters" element="ns:getRelationProcessHasAccountResponse" />
</wsdl:message>
<wsdl:message name="executeOPQLQueryRequest">
    <wsdl:part name="parameters" element="ns:executeOPQLQuery" />
</wsdl:message>
<wsdl:message name="executeOPQLQueryResponse">
    <wsdl:part name="parameters" element="ns:executeOPQLQueryResponse" />
</wsdl:message>
```

www.manaraa.com

```
<wsdl:portType name="OPQLPortType">
    <wsdl:operation name=
        "getRelationArtifactHasAccountWithOverlappedAccount">
        <wsdl:input message=
            "ns:getRelationArtifactHasAccountWithOverlappedAccountRequest"
            wsaw:Action=
            "urn:getRelationArtifactHasAccountWithOverlappedAccount" />
        <wsdl:output message=
            "ns:getRelationArtifactHasAccountWithOverlappedAccountResponse"
            wsaw:Action=
            "urn:getRelationArtifactHasAccountWithOverlappedAccountResponse" />
    </wsdl:operation>
    <wsdl:operation name="getRelationAgentHasAccount">
        <wsdl:input message="ns:getRelationAgentHasAccountRequest"
            wsaw:Action="urn:getRelationAgentHasAccount" />
        <wsdl:output message="ns:getRelationAgentHasAccountResponse"
            wsaw:Action="urn:getRelationAgentHasAccountResponse" />
    </wsdl:operation>
    <wsdl:operation name="getRelationProcessHasAccount">
        <wsdl:input message="ns:getRelationProcessHasAccountRequest"
            wsaw:Action="urn:getRelationProcessHasAccount" />
        <wsdl:output message="ns:getRelationProcessHasAccountResponse"
            wsaw:Action="urn:getRelationProcessHasAccountResponse" />
    </wsdl:operation>
```

www.manaraa.com

```
    <wsdl:operation name="executeOPQLQuery">
        <wsdl:input message="ns:executeOPQLQueryRequest"
            wsaw:Action="urn:executeOPQLQuery" />
        <wsdl:output message="ns:executeOPQLQueryResponse"
            wsaw:Action="urn:executeOPQLQueryResponse" />
    </wsdl:operation>
</wsdl:portType>
<wsdl:binding name="OPQLSoap11Binding" type="ns:OPQLPortType">
    <soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="document" />
    <wsdl:operation name="getRelationArtifactHasAccountWithOverlappedAccount">
        <soap:operation soapAction=
            "urn:getRelationArtifactHasAccountWithOverlappedAccount"
            style="document" />
        <wsdl:input>
            <soap:body use="literal" />
        </wsdl:input>
        <wsdl:output>
            <soap:body use="literal" />
        </wsdl:output>
    </wsdl:operation>
    <wsdl:operation name="getRelationAgentHasAccount">
        <soap:operation soapAction="urn:getRelationAgentHasAccount" style="document" />
        <wsdl:input>
            <soap:body use="literal" />
        </wsdl:input>
```

www.manaraa.com

```
        <wsdl:output>
            <soap:body use="literal" />
        </wsdl:output>
    </wsdl:operation>
    <wsdl:operation name="getRelationProcessHasAccount">
        <soap:operation soapAction="urn:getRelationProcessHasAccount"
            style="document" />
        <wsdl:input>
            <soap:body use="literal" />
        </wsdl:input>
        <wsdl:output>
            <soap:body use="literal" />
        </wsdl:output>
    </wsdl:operation>
    <wsdl:operation name="executeOPQLQuery">
        <soap:operation soapAction="urn:executeOPQLQuery" style="document" />
        <wsdl:input>
            <soap:body use="literal" />
        </wsdl:input>
        <wsdl:output>
            <soap:body use="literal" />
        </wsdl:output>
        </wsdl:operation>
</wsdl:binding>
```

www.manaraa.com

```
<wsdl:binding name="OPQLSoap12Binding" type="ns:OPQLPortType">
    <soap12:binding transport="http://schemas.xmlsoap.org/soap/http" style="document" />
    <wsdl:operation name="getRelationArtifactHasAccountWithOverlappedAccount">
        <soap12:operation soapAction=
            "urn:getRelationArtifactHasAccountWithOverlappedAccount"
            style="document" />
        <wsdl:input>
            <soap12:body use="literal" />
        </wsdl:input>
        <wsdl:output>
            <soap12:body use="literal" />
        </wsdl:output>
    </wsdl:operation>
    <wsdl:operation name="getRelationAgentHasAccount">
        <soap12:operation soapAction="urn:getRelationAgentHasAccount" style="document" />
        <wsdl:input>
            <soap12:body use="literal" />
        </wsdl:input>
        <wsdl:output>
            <soap12:body use="literal" />
        </wsdl:output>
    </wsdl:operation>
    <wsdl:operation name="getRelationProcessHasAccount">
        <soap12:operation soapAction="urn:getRelationProcessHasAccount" style="document" />
```

www.manaraa.com

```
        <wsdl:input>
            <soap12:body use="literal" />
        </wsdl:input>
        <wsdl:output>
            <soap12:body use="literal" />
        </wsdl:output>
    </wsdl:operation>
    <wsdl:operation name="executeOPQLQuery">
        <soap12:operation soapAction="urn:executeOPQLQuery" style="document" />
        <wsdl:input>
            <soap12:body use="literal" />
        </wsdl:input>
        <wsdl:output>
            <soap12:body use="literal" />
        </wsdl:output>
    </wsdl:operation>
</wsdl:binding>
<wsdl:binding name="OPQLHttpBinding" type="ns:OPQLPortType">
    <http:binding verb="POST" />
    <wsdl:operation name="getRelationArtifactHasAccountWithOverlappedAccount">
        <http:operation location=
            "OPQL/getRelationArtifactHasAccountWithOverlappedAccount" />
        <wsdl:input>
            <mime:content type="text/xml"
                part="getRelationArtifactHasAccountWithOverlappedAccount" />
```

www.manaraa.com

```
    </wsdl:input>
    <wsdl:output>
        <mime:content type="text/xml"
            part="getRelationArtifactHasAccountWithOverlappedAccount" />
    </wsdl:output>
</wsdl:operation>
<wsdl:operation name="getRelationAgentHasAccount">
    <http:operation location="OPQL/getRelationAgentHasAccount" />
    <wsdl:input>
        <mime:content type="text/xml" part="getRelationAgentHasAccount" />
    </wsdl:input>
    <wsdl:output>
        <mime:content type="text/xml" part="getRelationAgentHasAccount" />
    </wsdl:output>
</wsdl:operation>
<wsdl:operation name="getRelationProcessHasAccount">
    <http:operation location="OPQL/getRelationProcessHasAccount" />
    <wsdl:input>
        <mime:content type="text/xml" part="getRelationProcessHasAccount" />
    </wsdl:input>
    <wsdl:output>
        <mime:content type="text/xml" part="getRelationProcessHasAccount" />
    </wsdl:output>
</wsdl:operation>
```

www.manaraa.com

```
        <wsdl:operation name="executeOPQLQuery">
            <http:operation location="OPQL/executeOPQLQuery" />
            <wsdl:input>
                <mime:content type="text/xml" part="executeOPQLQuery" />
            </wsdl:input>
            <wsdl:output>
                <mime:content type="text/xml" part="executeOPQLQuery" />
            </wsdl:output>
        </wsdl:operation>
    </wsdl:binding>
    <wsdl:service name="OPQL">
        <wsdl:port name="OPQLHttpSoap11Endpoint" binding="ns:OPQLSoap11Binding">
            <soap:address location=
                "http://pm.viewsystem.org/axis2/services/OPQL.OPQLHttpSoap11Endpoint/" />
        </wsdl:port>
        <wsdl:port name="OPQLHttpSoap12Endpoint" binding="ns:OPQLSoap12Binding">
            <soap12:address location=
                "http://pm.viewsystem.org/axis2/services/OPQL.OPQLHttpSoap12Endpoint/" />
        </wsdl:port>
        <wsdl:port name="OPQLHttpEndpoint" binding="ns:OPQLHttpBinding">
            <http:address location=
                "http://pm.viewsystem.org/axis2/services/OPQL.OPQLHttpEndpoint/" />
        </wsdl:port>
    </wsdl:service>
</wsdl:definitions>
```

www.manaraa.com

BIBLIOGRAPHY

[1] E. Deelman and A. L. Chervenak, Data management challenges of data-intensive scientific workflows, *In Proc. of the International Symposium on Cluster Computing and the Grid (CCGRID)*, pages 687-692, 2008.

[2] S. B. Davidson and J. Freire, Provenance and scientific workflows: Challenges and opportunities, *In Proc. of the ACM SIGMOD International Conference on Management of Data (SIGMOD)*, pages 1345-1350, 2008.

[3] E. Deelman, Y. Gil, and M. Zemankova, *NSF Workshop on the Challenges of Scientific Workflows*, 2006.

[4] G. Bell, T. Hey, and A. Szalay, Beyond the data eeluge, *Science*, 323(6):1297-1298, 2009.

[5] C. Lin, S. Lu, X. Fei, A. Chebotko, D. Pai, Z. Lai, F. Fotouhi, and J. Hua, A reference architecture for scientific workflow management systems and the VIEW SOA solution, *IEEE Transactions on Services Computing (TSC)*, 2(1):79-92, 2009.

[6] C. Lin, S. Lu, Z. Lai, A. Chebotko, X. Fei, J. Hua, and F. Fotouhi, Service-oriented architecture for VIEW: A Visual Scientific Workflow Management System, *In Proc. of the IEEE International Conference on Services Computing (SCC)*, pages 335-342, 2008.

[7] A. Chebotko, S. Lu, X. Fei, and F. Fotouhi, RDFProv: A relational RDF store for querying and managing scientific workflow provenance, *Data & Knowledge Engineering (DKE)*, 69(8):836-865, 2010.

[8] P. Groth and L. Moreau, Recording process documentation for provenance, *IEEE Transactions on Parallel and Distributed Systems (TPDS)*, 20(9):1246-1259, 2009.

www.manaraa.com

[9] S. Miles, P. Groth, S. Munroe, S. Jiang, T. Assandri, and L. Moreau, Extracting causal graphs from an open provenance data model, *Concurrency and Computation: Practice and Experience*, 20(5):577-586, 2008.

[10] N. Kwasnikowska and J. V. Bussche, Mapping the NRC dataflow model to the Open Provenance Model, *In Proc. of the International Provenance and Annotation Workshop (IPAW)*, pages, 2008.

[11] D. L. McGuinness, J. Michaelis, and L. Moreau, Provenance and annotation of data and processes - Third International Provenance and Annotation Workshop (IPAW 2010), *Lecture Notes in Computer Science (LNCS)*, Vol. 6378, 2010.

[12] The Open Provenance Model (OPM) home page, `http://openprovenance.org`, 2007.

[13] The XML schema of the Open Provenance Model (v1.01), `http://openprovenance.org/model/v1.01.a`, 2008.

[14] The Provenance Challenge Wiki website, `http://twiki.ipaw.info/bin/view/Challenge/WebHome`, 2006.

[15] The Third Provenance Challenge Wiki website, `http://twiki.ipaw.info/bin/view/Challenge/ThirdProvenanceChallenge`, 2009.

[16] Pan-STARRS project home page, `http://ps1sc.org/`, 2009.

[17] L. Moreau (Editor), B. Plale, S. Miles, C. Goble, P. Missier, R. Barga, Y. Simmhan, J. Futrelle, R. E. McGrath, J. Myers, P. Paulson, S. Bowers, B. Ludäscher, N. Kwasnikowska, J. V. Bussche, T. Ellkvist, J. Freire, and P. Groth, The Open Provenance Model (v1.01), *Technical report*, University of Southampton, 2008.

[18] L. Moreau, B. Clifford, J. Freire, J. Futrelle, Y. Gil, P. Groth, N. Kwasnikowska, S. Miles, P. Missier, J. Myers, B. Plale, Y. Simmhan, E. Stephan, and J. V. Bussche, The Open

www.manaraa.com

Provenance Model core specification (v1.1), *Future Generation Computer Systems (FGCS)*, 27(6):743-756, 2011.

[19] Y. Simmhan, P. T. Groth, and L. Moreau, Special section: The Third Provenance Challenge on using the open provenance model for interoperability, *Future Generation Computer Systems (FGCS)*, 27(6):737-742, 2011.

[20] S. Chien, Z. Vagena, D. Zhang, V. J. Tsotras, and C. Zaniolo, Efficient structural joins on indexed XML documents, *In Proc. of the International Conference on Very Large Data Bases (VLDB)*, pages 263-274, 2002.

[21] B. He, Q. Luo, and B. Choi, Adaptive index utilization in memory-resident structural joins, *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 19(6):772-788, 2007.

[22] R. Jin, Y. Xiang, N. Ruan, and D. Fuhry, 3-HOP: A high-compression indexing scheme for reachability query, *In Proc. of the ACM SIGMOD International Conference on Management of Data (SIGMOD)*, pages 813-826, 2009.

[23] Y. Chen and Y. Chen, An efficient algorithm for answering graph reachability queries, *In Proc. of the International Conference on Data Engineering (ICDE)*, pages 893-902, 2008.

[24] S. P. Callahan, J. Freire, E. Santos, C. E. Scheidegger, C. T. Silva, and H. T. Vo, VisTrails: Visualization meets data management, *In Proc. of the ACM SIGMOD International Conference on Management of Data (SIGMOD)*, pages 745-747, 2006.

[25] C. E. Scheidegger, D. Koop, E. Santos, H. T. Vo, S. P. Callahan, J. Freire, and C. T. Silva, Tackling the Provenance Challenge one layer at a time, *Concurrency and Computation: Practice and Experience*, 20(5):473-483, 2008.

[26] C. Silva, J. Freire, and S. Callahan, Provenance for visualizations: Reproducibility and beyond, *IEEE Computing in Science and Engineering*, 9(5):82-29, 2007.

www.manaraa.com

[27] J. Freire, C. T. Silva, S. P. Callahan, E. Santos, C. E. Scheidegger, and H. T. Vo, Managing rapidly-evolving scientific workflows, *In Proc. of the International Provenance and Annotation Workshop (IPAW)*, pages 10-18, 2006.

[28] M. Anand, S. Bowers, and B. Ludäscher, Techniques for efficiently querying scientific workflow provenance graphs, *In Proc. of the International Conference on Extending Database Technology (EDBT)*, pages 287-298, 2010.

[29] M. Anand, S. Bowers, and B. Ludäscher, Provenance browser: Displaying and querying scientific workflow provenance graphs, *In Proc. of the IEEE International Conference on Data Engineering (ICDE)*, pages 1201-1204, 2010.

[30] M. Anand, S. Bowers, I. Altintas, and B. Ludäscher, Approaches for exploring and querying scientific workflow provenance graphs, *In Proc. of the International Provenance and Annotation Workshop (IPAW)*, pages 17-26, 2010.

[31] I. Altintas, M. Anand, D. Crawl, S. Bowers, A. Belloum, P. Missier, B. Ludäscher, C. Goble, and P. Sloot, Understanding collaborative studies through interoperable workflow provenance, *In Proc. of the International Provenance and Annotation Workshop (IPAW)*, pages 42-58, 2010.

[32] M. Anand, S. Bowers, and B. Ludäscher, A navigation model for exploring scientific workflow provenance graphs, *In Proc. of the Workshop on Workflows in Support of Large-Scale Science (SC-WORKS)*, 2009.

[33] M. Anand, S. Bowers, T. McPhillips, and B. Ludäscher, Exploring scientific workflow provenance using hybrid queries over nested data and lineage graphs, *In Proc. of the International Conference on Scientific and Statistical Database Management (SSDBM)*, pages 237-254, 2009.

[34] M. Anand, S. Bowers, T. McPhillips, and B. Ludäscher, Efficient provenance storage over nested data collections, *In Proc. of the International Conference on Extending Database Technology (EDBT)*, pages 958-969, 2009.

www.manaraa.com

[35] S. Bowers, T. McPhillips, S. Riddle, M. Anand, and B. Ludäscher, Kepler/pPOD: Scientific workflow and provenance support for assembling the tree of life, *In Proc. of the International Provenance and Annotation Workshop (IPAW)*, pages 70-77, 2008.

[36] S. Bowers, T. M. McPhillips, and B. Ludäscher, Provenance in collection-oriented scientific workflows, *Concurrency and Computation: Practice and Experience*, 20(5):519-529, 2008.

[37] I. Altintas, O. Barney, and E. Jaeger-Frank, Provenance collection support in the Kepler scientific workflow system, *In Proc. of the International Provenance and Annotation Workshop (IPAW)*, pages 118-132, 2006.

[38] S. Bowers, T. M. McPhillips, B. Ludäscher, S. Cohen, and S. B. Davidson, A model for user-oriented data provenance in pipelined scientific workflows, *In Proc. of the International Provenance and Annotation Workshop (IPAW)*, pages 133-147, 2006.

[39] S. B. Davidson, S. Khanna, S. Roy, J. Stoyanovich, V. Tannen, and Yi Chen, On provenance and privacy, *In Proc. of the IEEE International Conference on Database Theory (ICDT)*, pages 3-10, 2011.

[40] Z. Bao, S. B. Davidson, S. Khanna, and S. Roy, An optimal labeling scheme for workflow provenance using skeleton labels, *In Proc. of the ACM SIGMOD International Conference on Management of Data (SIGMOD)*, pages 711-722, 2010.

[41] S. B. Davidson, S. Khanna, D. Panigrahi, and S. Roy, Preserving module privacy in workflow provenance, *CoRR abs/1005.5543*, 2010.

[42] Z. Bao, S. C. Boulakia, S. B. Davidson, A. Eyal, and S. Khanna, Differencing provenance in scientific workflows, *In Proc. of the IEEE International Conference on Data Engineering (ICDE)*, pages 808-819, 2009.

[43] Z. Bao, S. C. Boulakia, S. B. Davidson, and P. Girard, PDiffView: Viewing the difference in provenance of workflow results, *In Proc. of the VLDB Endowment (PVLDB)*, 2(2): 1638-1641, 2009.

www.manaraa.com

[44] O. Biton, S. C. Boulakia, S. B. Davidson, and C. S. Hara, Querying and managing provenance through user views in scientific workflows, *In Proc. of the IEEE International Conference on Data Engineering (ICDE)*, pages 1072-1081, 2008.

[45] S. C. Boulakia, O. Biton, S. Cohen, and S. B. Davidson, Addressing the Provenance Challenge using ZOOM, *Concurrency and Computation: Practice and Experience*, 20(5): 497-506, 2008.

[46] O. Biton, S. C. Boulakia, and S. B. Davidson, Zoom*UserViews: Querying relevant provenance in workflow systems, *In Proc. of the International Conference on Very Large Data Bases (VLDB)*, pages 1366-1369, 2007.

[47] S. B. Davidson, S. C. Boulakia, A. Eyal, B. Ludäscher, T. M. McPhillips, S. Bowers, M. K. Anand, and J. Freire, Provenance in scientific workflow systems, *IEEE Data Engineering Bulletin (DEBU)*, 30(4): 44-50, 2007.

[48] S. Cohen, S. C. Boulakia, and S. B. Davidson, Towards a model of provenance and user views in scientific workflows, *In Proc. of the International Conference on Data Integration in the Life Sciences (DILS)*, pages 264-279, 2006.

[49] P. Missier, S. Sahoo, J. Zhao, C. Goble, and A. Sheth, Janus: From workflows to semantic provenance and linked open data, *In Proc. of the International Provenance and Annotation Workshop (IPAW)*, pages 129-141, 2010.

[50] P. Missier, N. W. Paton, and K. Belhajjame, Fine-grained and efficient lineage querying of collection-based workflow provenance, *In Proc. of the International Conference on Extending Database Technology (EDBT)*, pages 299-310, 2010.

[51] P. Missier, K. Belhajjame, J. Zhao, and C. Goble, Data lineage model for Taverna workflows with lightweight annotation requirements, *In Proc. of the International Provenance and Annotation Workshop (IPAW)*, pages 17-30, 2008.

www.manaraa.com

[52] A. Gibson, M. Gamble, K. Wolstencroft, T. Oinn, C. A. Goble, K. Belhajjame, and Paolo Missier, The data playground: An intuitive workflow specification environment, *Future Generation Computer Systems (FGCS)*, 25(4): 453-459, 2009.

[53] P. Missier, S. M. Embury, and R. Stapenhurst, Exploiting provenance to make sense of automated decisions in scientific workflows, *In Proc. of the International Provenance and Annotation Workshop (IPAW)*, pages 174-185, 2008.

[54] J. Zhao, C. Goble, R. Stevens, and D. Turi, Mining Taverna's Semantic Web of provenance, *Concurrency and Computation: Practice and Experience*, 20(5):463-472, 2008.

[55] A. Preece, P. Missier, S. Embury, B. Jin, and M. Greenwood, An ontology-based approach to handling information quality in e-Science, *Concurrency and Computation: Practice and Experience*, 20(3):253-264, 2008.

[56] P. Missier, P. Alper, O. Corcho, I. Dunlop, and C. Goble, Requirements and services for metadata management, *IEEE Internet Computing*, 11(5):17-25, 2007.

[57] D. Hull, K. Wolstencroft, R. Stevens, C. Goble, M. Pocock, P. Li, and T. Oinn, Taverna: A tool for building and running workflows of services, *Nucleic Acids Research (NAR)*, 34(Web-Server-Issue):729-732, 2006.

[58] R. S. Barga, Y. L. Simmhan, E. Chinthaka, S. S. Sahoo, J. Jackson, and N. Araujo, Provenance for scientific workflows towards reproducible research, *IEEE Data Engineering Bulletin (DEBU)*, 33(3):50-58, 2010.

[59] B. Cao, B. Plale, G. Subramanian, P. Missier, C. Goble, and Y. Simmhan, Semantically annotated provenance in the life science grid - The International Workshop on the role of Semantic Web in Provenance Management (SWPM 2009), *CEUR Workshop Proceedings (CEUR-WS.org)*, Vol. 526, 2009.

www.manaraa.com

[60] B. Cao, B. Plale, G. Subramanian, E. Robertson, and Y. Simmhan, Provenance information model of Karma version 3, *In Proc. of the IEEE International Workshop on Scientific Workflows (SWF)*, pages 348-351, 2009.

[61] Y. Simmhan, B. Plale, and D. Gannon, Karma2: Provenance management for data driven workflows, *International Journal of Web Services Research*, 5(2):1-22, 2008.

[62] Y. Simmhan, B. Plale, and D. Gannon, Query capabilities of the Karma provenance framework, *Concurrency and Computation: Practice and Experience*, 20(5):441-451, 2008.

[63] Y. Simmhan, B. Plale, and D. Gannon, A framework for collecting provenance in data-centric scientific workflows, *In Proc. of the IEEE International Conference on Web Services (ICWS)*, pages 427-436, 2006.

[64] Y. Simmhan, B. Plale, and D. Gannon, A survey of data provenance in e-Science, *ACM SIGMOD Record*, 34(3), 2005.

[65] D. Crawl and I. Altintas, A provenance-based fault tolerance mechanism for scientific workflows, *In Proc. of the International Provenance and Annotation Workshop (IPAW)*, pages 152-159, 2008.

[66] T. M. McPhillips, S. Bowers, D. Zinn, and B. Ludäscher, Scientific workflow design for mere mortals, *Future Generation Computer Systems (FGCS)*, 25(5):541-551, 2009.

[67] Y. Zhao, M. Hategan, B. Clifford, I. Foster, G. vonLaszewski, I. Raicu, T. Stef-Praun, and M. Wilde, Swift: Fast, reliable, loosely coupled parallel computation, *In Proc. of the IEEE International Workshop on Scientific Workflows (SWF)*, pages 199-206, 2007.

[68] Y. Zhao, M. Wilde, and I. T. Foster, Applying the virtual data provenance model, *In Proc. of the International Provenance and Annotation Workshop (IPAW)*, pages 148-161, 2006.

[69] Y. Zhao and S. Lu, A logic programming approach to scientific workflow provenance querying, *In Proc. of the International Provenance and Annotation Workshop (IPAW)*, pages 31-44, 2008.

www.manaraa.com

[70] M. Atay, A. Chebotko, D. Liu, S. Lu, and F. Fotouhi, Efficient schema-based XML-to-Relational data mapping, *Information Systems (IS)*, 32(3):458-476, 2007.

[71] A. Chebotko, C. Lin, X. Fei, Z. Lai, S. Lu, J. Hua, and F. Fotouhi, VIEW: A Visual Scientific Workflow Management System, *In Proc. of the IEEE International Workshop on Scientific Workflows (SWF)*, pages 207-208, 2007.

[72] A. Chebotko, X. Fei, C. Lin, S. Lu, and F. Fotouhi, Storing and querying scientific workflow provenance metadata using an RDBMS, *In Proc. of the IEEE International Conference on e-Science and Grid Computing (e-Science)*, pages 611-618, 2007.

[73] A. Chebotko, S. Chang, S. Lu, F. Fotouhi, and P. Yang, Scientific workflow provenance querying with security views, *In Proc. of the International Conference on Web-Age Information Management(WAIM)*, pages 349-356, 2008.

[74] L. Wang, S. Lu, X. Fei, A. Chebotko, H. Bryant, and J. Ram, Atomicity and provenance support for pipelined scientific workflows, *Future Generation Computer Systems (FGCS)*, 25(5):568-576, 2009.

[75] X. Fei, S. Lu, and C. Lin, A MapReduce-enabled scientific workflow composition framework, *In Proc. of the IEEE International Conference on Web Services (ICWS)*, pages 663-670, 2009.

[76] X. Fei and S. Lu, A collectional data model for scientific workflow composition, *In Proc. of the IEEE International Conference on Web Services (ICWS)*, pages 567-574, 2010.

[77] X. Fei and S. Lu, A dataflow-based scientific workflow composition framework, *IEEE Transactions on Services Computing (TSC)*, 2011. In press.

[78] J. Abraham, P. Brazier, A. Chebotko, J. Navarro, and A. Piazza, Distributed storage and querying techniques for a Semantic Web of scientific workflow provenance, *In Proc. of the IEEE International Conference on Services Computing (SCC)*, pages 178-185, 2010.

www.manaraa.com

[79] A. Chebotko, S. Lu, S. Chang, F. Fotouhi, and P. Yang, Secure abstraction views for scientific workflow provenance querying, *IEEE Transactions on Services Computing (TSC)*, 3(4):322-337, 2010.

[80] J. Alhiyafi, A scientific workflow system for genomic data analysis, *PhD thesis*, Wayne State University, 2010.

[81] C. Lim, S. Lu, A. Chebotko, and F. Fotouhi, Prospective and retrospective provenance collection in scientific workflow environments, *In Proc. of the IEEE International Conference on Services Computing (SCC)*, pages 449-456, 2010.

[82] C. Lim, S. Lu, A. Chebotko, and F. Fotouhi, Storing, reasoning, and querying OPM-compliant scientific workflow provenance using relational databases, *Future Generation Computer Systems (FGCS)*, 27(6):781-789, 2011.

[83] C. Lim, S. Lu, A. Chebotko, and F. Fotouhi, OPQL: A first OPM-level query language for scientific workflow provenance, *In Proc. of the IEEE International Conference on Services Computing (SCC)*, pages 136-143, 2011.

[84] P. Groth, S. Miles, and L. Moreau, A model of process documentation to determine provenance in mash-ups, *ACM Transactions on Internet Technology (TOIT)*, 9(1):1-31, 2009.

[85] P. T. Groth, S. Miles, W. Fang, S. C. Wong, K. Zauner, and L. Moreau, Recording and using provenance in a protein compressibility experiment, *In Proc. of the International Symposium on High Performance Distributed Computing (HPDC)*, pages 201-208, 2005.

[86] S. Miles, P. Groth, M. Branco, and L. Moreau, The requirements of recording and using provenance in e-science experiments, *Technical report*, University of Southampton, 2005.

[87] M. Szomszor and L. Moreau, Recording and reasoning over data provenance in Web and Grid services, *In Proc. of the International Conference on Ontologies, Databases, and Applications of Semantics (ODBASE)*, pages 603-620, 2003.

www.manaraa.com

[88] P. T. Groth, M. Luck, and L. Moreau, A protocol for recording provenance in service-oriented Grids, *In Proc. of the International Conference on Principles of Distributed Systems (OPODIS)*, pages 124-139, 2004.

[89] P. T. Groth, E. Deelman, G. Juve, G. Mehta, and G. B. Berriman, Pipeline-centric provenance model, *CoRR abs/1005.4457*, 2010.

[90] J. Kim, E. Deelman, Y. Gil, G. Mehta, and V. Ratnakar, Provenance trails in the Wings/Pegasus system, *Concurrency and Computation: Practice and Experience*, 20(5):587-597, 2008.

[91] D. A. Holland, U. Braun, D. Maclean, K. Muniswamy-Reddy, and M. Seltzer, Choosing a data model and query language for provenance, *In Proc. of the International Provenance and Annotation Workshop (IPAW)*, 2008.

[92] D. A. Holland, M. I. Seltzer, U. Braun, and K. Muniswamy-Reddy, PASSing the Provenance Challenge, *Concurrency and Computation: Practice and Experience*, 20(5):531-540, 2008.

[93] E. Deelman, D. Gannon, M. S. Shields, and I. Taylor, Workflows and e-Science: An overview of workflow system features and capabilities, *Future Generation Computer Systems (FGCS)*, 25(5):528-540, 2009.

[94] E. Elmroth, F. Hernández, and J. Tordsson, Three fundamental dimensions of scientific workflow interoperability: Model of computation, language, and execution environment, *Future Generation Computer Systems (FGCS)*, 26(2):245-256, 2010.

[95] A. Michlmayr, F. Rosenberg, P. Leitner, and S. Dustdar, Service provenance in QoS-aware Web service runtimes, *In Proc. of the IEEE International Conference on Web Services (ICWS)*, pages 115-122, 2009.

[96] W. Ding, J. Wang, and Y. Han, ViPen: A model supporting knowledge provenance for exploratory service composition, *In Proc. of the IEEE International Conference on Services Computing (SCC)*, pages 265-272, 2010.

www.manaraa.com

[97] S. Miles, P. T. Groth, M. Branco, and L. Moreau, The requirements of using provenance in e-Science experiments, *Journal of Grid Computing (GRID)*, 5(1):1-25, 2007.

[98] H. He and A. K. Singh, Graphs-at-a-time: query language and access methods for graph databases, *In the Proc. of the ACM SIGMOD International Conference on Management of Data (SIGMOD)*, pages 405-418, 2008.

[99] R. Zeng, X. He, and W.M.P. van der Aalst, A method to mine workflows from provenance for assisting scientific workflow composition, *In Proc. of the IEEE International Workshop on Scientific Workflows (SWF)*, pages 169-175, 2011.

[100] A. Michlmayr, F. Rosenberg, P. Leitner, and S. Dustdar, Selective service provenance in the VRESCo runtime, *International Journal of Web Service Research (IJWSR)*, 7(2):65-86, 2010.

[101] W. Tsai, X. Wei, Y. Chen, R. Paul, J. Chung, and D. Zhang, Data provenance in SOA: Security, reliability, and integrity, *Service Oriented Computing and Applications (SOCA)*, 1(4):223-247, 2007.

[102] S. Cruz, M. Campos, and M. Mattoso, Towards a taxonomy of provenance in scientific workflow management systems, *In Proc. of the IEEE International Workshop on Scientific Workflows (SWF)*, pages 259-266, 2009.

[103] T. Ellqvist, D. Koop, J. Freire, C. T. Silva, and L. Stromback, Using mediation to achieve provenance interoperability, *In Proc. of the IEEE International Workshop on Scientific Workflows (SWF)*, pages 291-298, 2009.

[104] A. Marinho, C. Werner, S. Cruz, M. Mattoso, V. Braganholo, and L. Murta, A strategy for provenance gathering in distributed scientific workflows, *In Proc. of the IEEE International Workshop on Scientific Workflows (SWF)*, pages 344-347, 2009.

www.manaraa.com

[105] I. Altintas, Lifecycle of scientific workflows and their provenance: A usage perspective, *In Proc. of the IEEE International Workshop on Scientific Workflows (SWF)*, pages 474-475, 2008.

[106] C. Ringelstein and S. Staab, DiALog: A distributed model for capturing provenance and auditing information, *International Journal of Web Service Research (IJWSR)*, 7(2):1-20, 2010.

[107] M. A. Vouk and M. P. Singh, Quality of service and scientific workflows, *In Proc. of the IFIP TC2/WG2.5 Working Conference on the Quality of Numerical Software*, pages 77-89, 1996.

[108] C. T. Silva, E. W. Anderson, E. Santos, and J. Freire, Using VisTrails and provenance for teaching scientific visualization, *Computer Graphic Forum*, 30(1):75-84, 2011.

[109] D. Zinn, Q. Hart, T. M. McPhillips, B. Ludäscher, Y. Simmhan, M. Giakkoupis, and V. K. Prasanna, Towards reliable, performant workflows for streaming-applications on cloud platforms, *In Proc. of the International Symposium on Cluster Computing and the Grid (CCGRID)*, pages 235-244, 2011.

[110] D. Yuan, Y. Yang, X. Liu, and J. Chen, On-demand minimum cost benchmarking for intermediate dataset storage in scientific cloud workflow systems, *Journal of Parallel and Distributed Computing (JPDC)*, 71(2):316-332, 2011.

[111] G. Juve and E. Deelman, Scientific workflows and clouds, *ACM Crossroads (CROSSROADS)*, 16(3):14-18, 2010.

[112] K. S. Shams, M. W. Powell, T. Crockett, J. S. Norris, R. Rossi, and T. Soderstrom, Polyphony: A workflow orchestration framework for cloud computing, *In Proc. of the International Symposium on Cluster Computing and the Grid (CCGRID)*, pages 606-611, 2010.

www.manaraa.com

[113] T. Huu and J. Montagnat, Virtual resources allocation for workflow-based applications distribution on a cloud infrastructure, *In Proc. of the International Symposium on Cluster Computing and the Grid (CCGRID)*, pages 612-617, 2010.

[114] D. Yuan, Y. Yang, X. Liu, and J. Chen, A data placement strategy in scientific cloud workflows, *Future Generation Computer Systems (FGCS)*, 26(8):1200-1214, 2010.

[115] P. T. Groth and L. Moreau, Representing distributed systems using the Open Provenance Model, *Future Generation Computer Systems (FGCS)*, 27(6):757-765, 2011.

[116] D. Yuan, Y. Yang, X. Liu, and J. Chen, A cost-effective strategy for intermediate data storage in scientific cloud workflow systems, *In Proc. of the IEEE International Parallel & Distributed Processing Symposium (IPDPS)*, pages 1-12, 2011.

[117] C. Zhang and H. Sterck, CloudWF: A computational workflow system for clouds based on Hadoop, *In Proc. of the IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*, pages 393-404, 2009.

[118] S. B. Davidson, S. Khanna, V. Tannen, S. Roy, Y. Chen, T. Milo, and J. Stoyanovich, Enabling privacy in provenance-aware workflow systems, *In Proc. of the biennial Conference on Innovative Data Systems Research (CIDR)*, pages 215-218, 2011.

[119] P. Buneman, S. Khanna, and W. C. Tan, Why and where: A characterization of data provenance, *In Proc. of the International Conference on Database Theory (ICDT)*, pages 316-330, 2001.

[120] P. Buneman, S. Khanna, and W. C. Tan, Data provenance: Some basic issues, *In Proc. of the International Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS)*, pages 87-93, 2000.

[121] P. Buneman and W. C. Tan, Provenance in databases, *In Proc. of the ACM SIGMOD International Conference on Management of Data (SIGMOD)*, pages 1171-1173, 2007.

www.manaraa.com

[122] Olaf Hartig, Provenance information in the Web of data, *In Proc. of the International Workshop on Linked Data on the Web (LDOW)*, 2009.

[123] Y. Gil and P. T. Groth, Using provenance in the Semantic Web, *Journal of Web Semantics (WS)*, 9(2):147-148, 2011.

www.manaraa.com

ABSTRACT

QUERYING AND MANAGING OPM-COMPLIANT SCIENTIFIC WORKFLOW PROVENANCE

by

CHUNHYEOK LIM

December 2011

Advisor: Dr. Farshad Fotouhi

Co-advisor: Dr. Shiyong Lu

Major: Computer Science

Degree: Doctor of Philosophy

Provenance, the metadata that records the derivation history of scientific results, is important in scientific workflows to interpret, validate, and analyze the result of scientific computing. Recently, to promote and facilitate provenance interoperability among heterogeneous provenance systems, the Open Provenance Model (OPM) has been proposed and has played an important role in the community. In this dissertation, to efficiently query and manage OPM-compliant provenance, we first propose a provenance collection framework that collects both prospective provenance, which captures an abstract workflow specification as a recipe for future data derivation and retrospective provenance, which captures past workflow execution and data derivation information. We then propose a relational database-based provenance system, called OPMPROV that stores, reasons, and queries prospective provenance and retrospective provenance, which is OPM-compliant provenance. We finally propose *OPQL*, an OPM-level provenance query language, that is directly defined over the OPM model. An *OPQL* query takes an OPM graph as input and produces an OPM graph as output; therefore, *OPQL* queries are not tightly coupled to the underlying provenance storage strategies. Our provenance collection framework, provenance store, and provenance query language feature the native support of the OPM model.

www.manaraa.com

AUTOBIOGRAPHICAL STATEMENT

CHUNHYEOK LIM

EDUCATION

- Doctor of Philosophy (Computer Science), December 2011
  Wayne State University, Detroit, Michigan, United States

- Master of Science (Computer Science), January 2001
  Korean National Defense University, Seoul, Republic of Korea

- Bachelor of Science (Computer Science), February 1996
  Korea Military Academy, Seoul, Republic of Korea

PUBLICATIONS

- **Chunhyeok Lim**, Shiyong Lu, Artem Chebotko, and Farshad Fotouhi, *"Prospective and Retrospective Provenance Collection in Scientific Workflow Environments",* In Proc. of the IEEE International Conference on Services Computing (SCC), pages 449-456, Miami, Florida, USA, July 2010.

- **Chunhyeok Lim**, Shiyong Lu, Artem Chebotko, and Farshad Fotouhi, *"Storing, reasoning, and querying OPM-compliant scientific workflow provenance using relational databases",* Future Generation Computer Systems (FGCS), 27(6):781-789, 2011.

- **Chunhyeok Lim**, Shiyong Lu, Artem Chebotko, and Farshad Fotouhi, *"OPQL: A First OPM-Level Query Language for Scientific Workflow Provenance",* In Proc. of the IEEE International Conference on Services Computing (SCC), pages 136-143, Washington, D.C., USA, July 2011.

- **Chunhyeok Lim**, Shiyong Lu, Artem Chebotko, and Farshad Fotouhi, *"A First OPM-Compliant Provenance Service for Scientific Workflows",* IEEE Transactions on Services Computing (TSC), 2011. (submitted).

AWARDS AND HONORS

- Overseas Education Scholarship for Three Years (August 2007 - August 2010), Republic of Korea Army, August 2007.

www.manaraa.com