12-2008

Butterfly : A Provenance Management System

Yaobin Tang
Worcester Polytechnic Institute

Murali Mani
Worcester Polytechnic Institute, mmani@cs.wpi.edu

WPI-CS-TR-08-16 December 2008

Butterfly : A Provenance Management System

by

Yaobin Tang
Murali Mani

Butterfly : A Provenance Management System

Yaobin Tang, Murali Mani
Computer Science Department, WPI

Abstract

Semantically rich metadata is foreseen to be pervasive in tomorrow's cyber world. People are more willing to store metadata in the hope that such extra information will enable a wide range of novel business intelligent applications. Provenance is metadata which describes the derivation history of data. It is considered to have great potential for helping the reasoning, analyzing, validating, monitoring, integrating and reusing of data. In this paper, we introduce Butterfly, a provenance management system, which offers the modeling, storage, and query of provenance.

1 Motivation

With today's abundant computer storage and powerful processing capability, people become more and more aggressive in collecting extra data: data intentionally generated to assist the understanding of other data or processes. Simple form of model and query of such data can not satisfy non-expert users' growing appetite for intelligent support in applications. For example,

- an online catalog vendor wants to track the interaction of customers with the UI to discover the sequential pattern of operations which ends in purchasing a product; find the most visited (used) web page (interface) to improve user experience; query the connection between two visited web pages to better understand user behavior and enhance cross-selling.
- a scientist wants to log detailed running steps and intermediate results of an experiment saving the opportunity for future inspection or reproduction of the result; providing poof to peer scientists about the authenticity of the experiment; contributing to the pool of experimental recipes for reuse.
- a food manufacturer wants to record the production and distribution process of a product, so whenever the product is found flawed, it is possible to trace back to the origin of the problem; map the affected vendors, shops, and regions; estimate the ensuing loss and compensation; submit report to supervisory authority for conformity check.

There are some common patterns in the aforementioned scenarios:

- People are interested in collecting transient ancillary information, which is usually not captured or simply discarded. The form of such information is diverse varying with domain.
- Ad hoc queries are asked about complex relationship among data. Answers to these queries can benefit people with insight into the domain.

Having realized the usefulness of ancillary data and having envisioned its popularity, we have launched the `MetaWare` project, which aims at providing a general solution to the management of metadata.

2 Introduction

We are particularly interested in provenance, a special kind of metadata that assists in understanding how things are *related* to each other in their derivation history. Inquiry of provenance is pervasive in everyday life, and a lot of applications (like the previous examples) profit from the ability to know the provenance of an entity.

However, two obstacles are in the way of managing provenance:

- Before we can store provenance, we need a way to describe them.
- Once provenance is stored, we need a query language to extract information from them.

Our solution to the first problem is a *semantic model* of provenance. Our solution to the second problem is an algebraic *query model*. We combine these two solutions and call it `Butterfly` (named after the butterfly effect). It is a virtual proof-of-concept provenance management system. We are also working on an initial implementation of `Butterfly`, which actualizes the semantic model with a provenance definition language, and actualizes the query model with a provenance query language.

Assumptions We believe a provenance management system should be agnostic of application domains. We argue that the system itself should not be held responsible for capturing provenance. Instead, users should decide on what and how they capture provenance according to their particular needs and views.

Usage Scenario

We imagine a typical scenario of applying `Butterfly` would look like this: A programmer is assigned to build a provenance-aware application for a hospital (e.g., Electronic Health Record or simply paperless office application). He knows there is a handy middleware called `Butterfly` which he can easily integrate into his application for processing provenance. What he needs to do is embed some provenance processing codes into appropriate places of the business logic. He can decide what interesting events to record and how to describe them (e.g., granularity) using the semantic model. The provenance processing codes talk to `Butterfly` in the definition language. After the host application has run several months, a patient revisits the hospital. A doctor opens an editor and composes some queries to pull out the sequence of treatment the patient has received in the past. Later, another patient files a complaint. A supervisor queries the provenance base to investigate if there is any violation of regulation in the sequence of treatment.

Contributions

Although there are many systems that utilize the idea of keeping track of data provenance, most of their provenance tracking components are highly coupled to their application logic (less reusable) and their provided queries are simple. More detailed discussion of related work is given in section 6. Standardization of a general purpose provenance model has recently aroused attention from the provenance research community. For example, [11] proposes the Open Provenance Model (OPM). The fundamental difference between OPM and our work is mainly trifold: *a*) OPM classifies causal relationship into five exact relationships. We use a loose, uniform relationship and leave the exact interpretation of relationship to varied domains. *b*) OPM queries are based on logical inference from the five causal relationships. We use an algebraic approach instead. *c*) Our goal is to provide a *general purpose, independent, working* management system (along with APIs) for provenance. In order to do that, we specify the structural representation of each provenance concept.

Roadmap The rest of this paper is structured as follows: we introduce the semantic model in section 3, the query model in section 4, the system architecture in section 5. We discuss related work in section 6, and conclude with section 7.

3 Semantic Model of Provenance

Definition 3.0.1: A *name space* is an infinite countable set, denoted as $\aleph$.
e.g., the set of all possible ASCII strings is a name space.
Definition 3.0.2: A *named value* is $(name, value\text{-}list)$. $name \in \aleph$. *value-list* is a list, and it can be empty.
e.g., (Buyer, (John Green)) is a named value. Note if *value-list* contains only one element, we can omit the parentheses of *value-list*. e.g., (Money, \$40k) is also a named value.
Definition 3.0.3: An *identifier* is a named value.
Two identifiers are equal if and only if their *names* are equal. For example, (R08, (Data File MedReport)) is an identifier, and is equal to (R08, ()).

The semantic model provides a tool for describing provenance. Particularly, it defines several essential concepts that we believe are closely related to the modeling of provenance. Our goal in this section is to define what are *provenance entity* and *provenance relationship*.

3.1 Application Environment

Definition 3.1: An *application environment* is

$$(\mathit{NSpace}, \mathcal{CLK}, \mathcal{ADDR}, \mathcal{DICT}).$$

NSpace is a name space. $\mathcal{CLK}$ is a set of system-recognizable time (e.g., in the format of MM-DD-YY). $\mathcal{ADDR}$ is a set of system-recognizable addresses (e.g., URI address). $\mathcal{DICT} \subseteq \mathit{NSpace}$, is a defined vocabulary.

Application environment mandates a minimum integrity constraint on the data, and provides some degree of interoperability. Note the following concepts are all defined with respect to an environment.

3.2 Annotation

Definition 3.2: An *annotation* is a set of named values. Each named value (*name*, *value-list*) is called an *entry*, and $\mathit{name} \in \mathcal{DICT}$.
e.g., {(total-income, \$40k), (mortgage-limit, \$100k), (mortgage-type, fixed-rate)} could be an annotation of a mortgage application.

Annotation provides an extensible way to describe something. Due to diversity of domains, we impose only little restriction upon annotation.

3.3 Static Entity Record

Defintion 3.3: A *static entity record* is

$$(\textit{entity-id}, \textit{entity-address}, \textit{entity-type}, \textit{annotation}, \\ \textit{snapshot-time}, \textit{record-id}).$$

entity-id is an identifier. $\textit{entity-address} \in \mathcal{ADDR}$. $\textit{entity-type} \in \mathcal{DICT}$. *annotation* is an annotation as in Def. 3.2. $\textit{snapshot-time} \in \mathcal{CLK}$. $\textit{record-id} \in \mathit{NSpace}$.

Static entity record captures some *aspects* of an entity at a particular *moment* (*snapshot-time*). e.g,

```
entity-id=(S01, (Aspect, ···))
entity-address=100 Inst. Rd
entity-type=Human
annotation={(Income, $40k), ···}
snapshot-time=11-01-07
record-id=R01
```

is a static entity record capturing some facts about a person at some moment. Note there is nontrivial distinction between *entity* and *static entity record*. The former one refers to the evolving physical existence, while the latter one refers to a virtual representation (as a record) of some aspects of that entity at a particular moment. With that being mentioned, it is clear that: *a*) Our system does not intend to manipulate (e.g., store) the actual entities (e.g., data files, pictures), but our system will manipulate their corresponding representations. *b*) Theoretically, static entity record is about some facts in the past, and should not be updated.

entity-id, *entity-address*, *entity-type* all refer to an *entity*. They are self-explanatory. One example of *entity-address* is Uniform Resource Identifier (URI). Currently, *entity-type* is simply drawn from $\mathcal{DICT}$. We intend to introduce hierarchy of entity types in the future for richer modeling. Past facts (states) of an entity can be amassed by grouping according to *entity-id* (recall the equality definition of identifier). Static entity record with the latest *snapshot-time* represents the closest approximation of the corresponding entity. *record-id* uniquely identifies the *record* itself not the *entity*.

3.4 Activity Record

Definition 3.4.1: An *activity type* is

$$(\textit{type-name}, \textit{incoming-list}, \textit{outgoing-list}, \textit{annotation}).$$

type-name is an identifier. *incoming-list* is an ordered list of named values (each named value in the *incoming-list* is called an *in-pipe*). *outgoing-list* is an ordered list of named values (each named value in the *outgoing-list* is called an *out-pipe*). *annotation* is an annotation as in Def. 3.2.

incoming-list declares the *roles* and *types* of the *contributing* entities in an activity. For example,

((Doctor, Human) (Patient, Human) (X-Ray, Machine))

is an *incoming-list*, with "Doctor", "Patient", "X-Ray" as roles and "Human", "Machine" as types. Types should be taken from $\mathcal{DICT}$.

Similarly, *outgoing-list* specifies the roles and types of the *consequential* entities in the activity. For example, ((Radiographic-Image, Image) (Diagnosis, Report))

is an *outgoing-list*. The types "Image" and "Report" should be taken from $\mathcal{DICT}$ as well.

Define a symbol $\underline{\text{ACT}}$, which denotes a dummy activity type. It symbolizes an insignificant or unknown activity.

Flexibility Because our model is meant to be simple but still flexible and capable, in our system, we avoid forced detail modeling of activity but reserve the potential for doing that. For example, based on *grouping* by *type-name*, we can support the concepts of hierarchical view and equivalent view of activities as follows: Recall *type-name* (as in Def. 3.4.1) is an identifier, with the form of (*name*, *value-list*). Let *value-list* = (*event*, *generality*, *explanation*). Intuitively, *event* clusters relevant activities of an event. Activities with a smaller *generality* value offer a more detailed view of the event. When *generality* values are the same, activities with a smaller *explanation* value are considered a more plausible explanation of the event. Figure 1 shows an example. It shows there is a more detailed view of "Building Caught Fire". The fire is more likely a result of lightning (*explanation*=1) than short-circuiting (*explanation*=2). There is also a more detailed view of "Firefighters Put Out Fire".

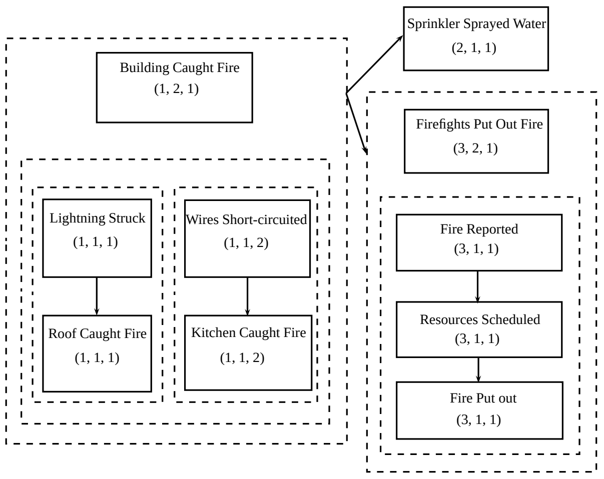


Figure 1: Flexibility—Grouping of Activities

Definition 3.4.2: An *activity record* is

$$(\textit{activity-id}, \textit{activity-type}, \textit{activity-span}, \textit{annotation}, \textit{record-id}).$$

activity-id is an identifier. *activity-type* is an activity type as in Def. 3.4.1. *activity-span*=(*start*, *end*), *start* and *end* $\in \mathcal{CLK}$. *annotation* is an annotation as in Def. 3.2. *record-id* $\in$ *NSpace*.

In contrast to static entity record, activity record is used to describe any *dynamic* element of an application system. It is an instance of an activity type, and a *representation* of an activity taking place in the physical world (e.g., an invocation of a computing function).

Each component of activity record is self-explanatory. We recommend to store in *annotation* additional application specific information about the running of an activity.

3.5 Provenance Entity And Relationship

A **provenance entity** is either a static entity record (Def. 3.3) or an activity record (Def. 3.4.2).

Definition 3.5.1: A **provenance relationship** is a relationship between two provenance entities:

$$(\textit{causal-entity}, \textit{consequential-entity}, \textit{role}, \textit{annotation}, \textit{relationship-id}).$$

causal-entity, *consequential-entity* are both provenance entities. *role* $\in \mathcal{DICT}$. *annotation* is an annotation as in Def. 3.2. *relationship-id* $\in$ *NSpace*.

Relationship between *causal-entity* and *consequential-entity* can be thought of as a parent-child relationship. *role* refines the relationship by supplementing the role that *causal-entity* played in the creation of *consequential-entity*. Compatibility check is required. e.g., when *causal-entity* is a static entity record and *consequential-entity* is an activity record, *role* and *causal-entity* must be meaningful to *consequential-entity* (e.g., in Figure 2, the "buying" role and "Buyer" type match an *in-pipe* of *activity-type* "Closing").

The semantic model provides a flexible way to describe provenance of both static and dynamic elements of an application system. Figure 2 shows a simplified example of house mortgage, where a prospective buyer got a house offer through a real estate broker, applied mortgage from a mortgage company, closed the transaction with the seller and got a new house title. There are two activities, one of which is of dummy type and the other is of "Closing" type. Provenance relationship is shown as directed link from *causal-entity* to *consequential-entity*, with *role* as the label of the link, *annotation* and *relationship-id* omitted for simplicity.

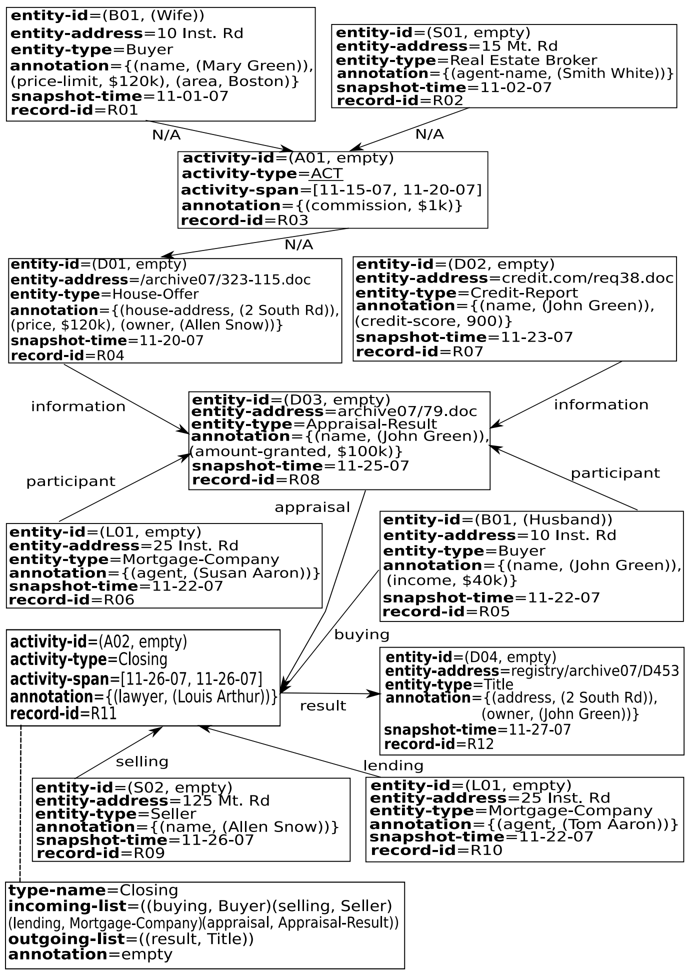

entity-id=(B01, (Wife))
entity-address=10 Inst. Rd
entity-type=Buyer
annotation={(name, (Mary Green)), (price-limit, $120k), (area, Boston))}
snapshot-time=11-01-07
record-id=R01
entity-id=(S01, empty)
entity-address=15 Mt. Rd
entity-type=Real Estate Broker
annotation={(agent-name, (Smith White))}
snapshot-time=11-02-07
record-id=R02
N/A
N/A
activity-id=(A01, empty)
activity-type=ACT
activity-span=[11-15-07, 11-20-07]
annotation={(commission, $1k)}
record-id=R03
N/A
entity-id=(D01, empty)
entity-address=/archive07/323-115.doc
entity-type=House-Offer
annotation={(house-address, (2 South Rd)), (price, $120k), (owner, (Allen Snow))}
snapshot-time=11-20-07
record-id=R04
entity-id=(D02, empty)
entity-address=credit.com/req38.doc
entity-type=Credit-Report
annotation={(name, (John Green)), (credit-score, 900)}
snapshot-time=11-23-07
record-id=R07
information
information
entity-id=(D03, empty)
entity-address=archive07/79.doc
entity-type=Appraisal-Result
annotation={(name, (John Green)), (amount-granted, $100k)}
snapshot-time=11-25-07
record-id=R08
participant
participant
appraisal
entity-id=(L01, empty)
entity-address=25 Inst. Rd
entity-type=Mortgage-Company
annotation={(agent, (Susan Aaron))}
snapshot-time=11-22-07
record-id=R06
entity-id=(B01, (Husband))
entity-address=10 Inst. Rd
entity-type=Buyer
annotation={(name, (John Green)), (income, $40k)}
snapshot-time=11-22-07
record-id=R05
buying
activity-id=(A02, empty)
activity-type=Closing
activity-span=[11-26-07, 11-26-07]
annotation={(lawyer, (Louis Arthur))}
record-id=R11
result
entity-id=(D04, empty)
entity-address=registry/archive07/D453
entity-type=Title
annotation={(address, (2 South Rd)), (owner, (John Green))}
snapshot-time=11-27-07
record-id=R12
selling
lending
entity-id=(S02, empty)
entity-address=125 Mt. Rd
entity-type=Seller
annotation={(name, (Allen Snow))}
snapshot-time=11-26-07
record-id=R09
entity-id=(L01, empty)
entity-address=25 Inst. Rd
entity-type=Mortgage-Company
annotation={(agent, (Tom Aaron))}
snapshot-time=11-22-07
record-id=R10
type-name=Closing
incoming-list=((buying, Buyer)(selling, Seller)(lending, Mortgage-Company)(appraisal, Appraisal-Result))
outgoing-list=((result, Title))
annotation=empty

Figure 2: Example—Semantic Model of Provenance

4 Query Model of Provenance

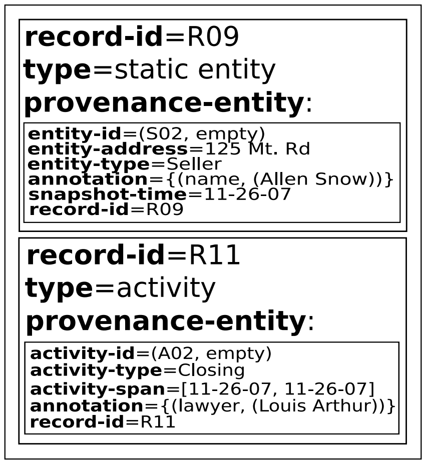
We have defined the semantics of provenance entity and provenance relationship in section 3. In this section, our goal is to develop an algebraic query model to manipulate provenance entity and relationship. This query model bases on two sub-models: a *content based model* for content based query of provenance and a *structure based model* for structure based query of provenance. By combining the power of two sub-models, we can express a lot of interesting provenance queries. In order to define the content based query model, we need to first develop two concepts: *provenance entity store* and *provenance relationship store*. They are corresponding to provenance entity and provenance relationship in the semantic model.

4.1 Provenance Entity Store

Definition 4.1.1: A *provenance entity store* is a set of provenance entities. Recall from section 3, a provenance entity is either a static entity record (definition 3.3) or an activity record (definition 3.4.2). Since each provenance entity is uniquely identified by its *record-id*, we can have a representation schema for provenance entity store as follows:

$$(\textit{record-id}, \textit{type}, \textit{provenance-entity}).$$

type can either be "static entity" or "activity". *provenance-entity* holds the content of the provenance entity. Here is an example of provenance entity store:



record-id=R09
type=static entity
provenance-entity:

entity-id=(S02, empty)
entity-address=125 Mt. Rd
entity-type=Seller
annotation={(name, (Allen Snow))}
snapshot-time=11-26-07
record-id=R09

record-id=R11
type=activity
provenance-entity:

activity-id=(A02, empty)
activity-type=Closing
activity-span=[11-26-07, 11-26-07]
annotation={(lawyer, (Louis Arthur))}
record-id=R11

Operators For Provenance Entity Store

We can define several manipulating operators for provenance entity store. The following set of operators is not intended to be exhaustive.

Operator 4.1.1 (filtering by type): The *filtering-by-type* operator takes a provenance entity store and a *type* as input, and produces a provenance entity store with all and only provenance entities of the *type*.
e.g., we can use this operator to retrieve all activity records in a provenance entity store.

Defintion 4.1.2: A *conditional function* of provenance entity takes a provenance entity and a list of parameters as input, and produces either "True" or "False".
Here is a simple example:

```
BOOLEAN
TimeLargerThan(ProvenanceEntity e, CLK time)
  IF (e.SnapshotTime > time) RETURN True
  RETURN False
```

Conditional functions are combinable using logical conjunctions (e.g., "And", "Or", "Not") to achieve more expressiveness. Still, conditional functions could be very complex and highly native to an application domain (e.g., in Figure 1, a condition like "an activity that belongs to the most plausible and detailed view of an event"). Thus besides providing basic conditional functions (e.g., equality, greater than, less than, pattern matching test) for each component of provenance entity (e.g., *entity-address* in static entity record, *activity-span* in activity record), we need to support user customized conditional functions that comply with the function signature in definition 4.1.2. Detailed discussion of the design of conditional functions is beyond the scope of this paper. We simply consider it as an abstraction here.

Operator 4.1.2 (filtering by condition): The *filtering-by-condition* operator takes a provenance entity store and a condition (conditional function plus list of actual parameters) as input, and produces a provenance entity store with all and only provenance entities that satisfy the condition.
e.g., we can use the filtering-by-condition operator to retrieve all activity records that happened within the time interval [10-01-07, 12-1-07] and were annotated as having "critical-level" larger than 99.

Operator 4.1.3 (set operators): Since provenance entity store is defined as a set of provenance entities, ordinary set operators (e.g., union, intersection, difference) are applicable to provenance entity store as well.

The next two operators are mainly used for interfacing with the structure based sub-model of query.

Operator 4.1.4 (filtering by record-id): The *filtering-by-record-id* operator takes a provenance entity store and a set of *record-ids* as input, and produces a provenance entity store with all and only provenance entities whose *record-ids* are in the *record-id* set.

Operator 4.1.5 (projecting of record-id):
The *projecting-record-id* operator takes a provenance entity store as input, and produces a set of *record-ids* of all the provenance entities in the store.

Definition 4.1.3 (valuation): A *valuation function* of provenance entity takes a provenance entity as input, and produces a value as output. A *value adder* is a function $\oplus : \mathbb{VALUE} \times \mathbb{VALUE} \rightarrow \mathbb{VALUE}$.
We do not give an explicit meaning for "value" here. It could be a number, string etc depending on applications. Binary arithmetic or maximum operators, string concatenation operator are examples of value adder.

Operator 4.1.6 (aggregation): The *aggregation* operator takes a provenance entity store, a valuation function and a value adder as input, and produces a value by applying the valuation function to each provenance entity and combining the values using the value adder.

Except for operators 4.1.5 and 4.1.6, provenance entity store is closed under the others. We can compose those operators to form a query.

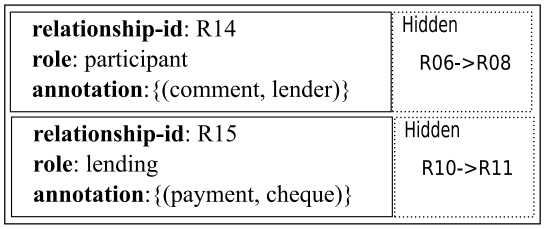
4.2 Provenance Relationship Store

Similar to the concept of provenance entity store, we can define the concept of provenance relationship store.

Definition 4.2.1: A *provenance relationship store* is a set of provenance relationships.
However, in provenance relationship store, we only perceive each provenance relationship as

$$(relationship\text{-}id, role, annotation).$$

Compared with definition 3.5.1 of provenance relationship, we hide *causal-entity* and *consequential-entity* from provenance relationship. This is because, in the content based query model, we focus on the *content* (i.e. *role* and *annotation*) of the relationship instead of its *structure* (i.e. from *causal-entity* to *consequential-entity*). Here is an example of provenance relationship store (referring to Figure 2):



relationship-id: R14 **role**: participant **annotation**:{(comment, lender)}	Hidden R06->R08
relationship-id: R15 **role**: lending **annotation**:{(payment, cheque)}	Hidden R10->R11

Operators For Provenance Relationship Store

Provenance entity store and relationship store are very similar (e.g., both defined as set, both aimed for content style query). We can define a rather similar set of operators (e.g., filtering, aggregating, interfacing) for manipulating relationship store. The following examples are not intended to be exhaustive.

Operator 4.2.1 (filtering by role): The *filtering-by-role* operator takes a provenance relationship store and a *role* as input, and produces a provenance relationship store with all and only provenance relationships that have the specified

role.
e.g., we can use this operator to retrieve all provenance relationships of the "lending" type.

Operator 4.2.2 (filtering by annotation): The *filtering-by-annotation* operator takes a provenance relationship store and an annotation condition as input, and produces a provenance relationship store with all and only provenance relationships that satisfy the condition.
e.g., imagine in an application, each provenance relationship was annotated with (*importance*, *number*) to indicate how important a role had been. Then we can retrieve all the provenance relationships with *importance* above a threshold.

Operator 4.2.3 (set operators): Ordinary set operators (e.g., union, intersection, difference) are applicable to provenance relationship store.

Operator 4.2.4 (aggregation): The aggregation operator takes a provenance relationship store, a valuation function of annotation and a value adder as input, and produces a value by applying the valuation function to the annotation of each provenance relationship and combining the values using the value adder.
e.g., if a valuation function maps any annotation to 1, by using this operator, we can count the number of provenance relationships in a store.

In order to interface provenance relationship store and structure based query model, we need the next two operators.

Operator 4.2.5 (filtering by relationship-id):
The *filtering-by-relationship-id* operator takes a provenance relationship store and a set of *relationship-ids* as input, and produces a provenance relationship store with all and only provenance relationships whose *relationship-ids* are in the *relationship-id* set.

Operator 4.2.6 (projecting of relationship-id):
The *projecting-relationship-id* operator takes a provenance relationship store as input, and produces a set of *relationship-ids* of all the provenance relationships in the store.

4.3 Content Based Query Model

Definition 4.3: The concept of *provenance entity store* and its relevant query operators and the concept of *provenance relationship store* and its relevant query operators constitute the *content based query model* of provenance.

The distinctiveness of provenance data compared with ordinary record data resides in its implication of complex relationships. For ordinary record data, the main goal of query design is being able to retrieve the content of data (content-oriented). However, for provenance data, it is also important to understand the underlying complex relationships (structure-oriented). Because of that, we divide the query model of provenance into two sub-models: one for content based query and one for structure based query. A provenance query seamlessly integrates these two sub-models.

Next, we introduce the structure based model of query.

4.4 Structure Based Query Model

Definition 4.4.1: A *structure graph* of provenance is

$$(N, E, f).$$

$N \subseteq \mathit{NSpace}$, is a set of *record-ids* of provenance entities. $E \subseteq \mathit{NSpace}$, is a set of *relationship-ids* of provenance relationships. $f : E \to N \times N$ is a function that specifies the *causal-entity* and *consequential-entity* of a relationship.

To perform query over structure graphs, we need to define a set of manipulating operators. A few operators are illustrated here. It should be emphasized that users may want to invent their own sets of operators which suit particular application needs. So it is important that users are able to implement their own operators and plug them into the structure model framework. The following operators are intended to be informative rather than exhaustive.

Ordinary Operators

The next two operators are mainly for interfacing with content based query model.

Operator 4.4.1 (projecting of record-id):
The *projecting-of-record-id* operator takes a structure graph (N, E, f) as input, and produces N as output.

Operator 4.4.2 (projecting of relationship-id):
The *projecting-of-relationship-id* operator takes a structure graph (N, E, f) as input, and produces E as output.

Sometimes, users want to narrow the structure graph to focus on some interesting entities and their relationships. If we perceive $f : E \to N \times N$ as a function that defines how "edges" connect "nodes", we can abstract the user requirements as follows:

Operator 4.4.3 (reducing by record-id):
The *reducing-by-record-id* operator takes a structure graph (N, E, f) and a set N' of *record-ids* as input, and produces a structure graph with $\{n \mid \exists a, b \in N \cap N', n \in N \text{ is on a path from } a \text{ to } b\} \cup (N \cap N')$ as the set of *record-ids*, and

$\{e \mid \exists a, b \in N \cap N', e \in E$ is on a path from a to $b\}$ as the set of *relationship-ids*.

Operator 4.4.4 (abstracting by record-id):
The *abstracting-by-record-id* operator takes a structure graph (N, E, f) and a set N' of *record-ids* as input, and produces a structure graph with $N \cap N'$ as the set of *record-ids*, and the set of *relationship-ids* defined as follows: Let C_0=$\{(a, b, 0) \mid a, b \in N \cap N', \exists e \in E, f(e) = (a, b)\}$, C_1=$\{(a, b, 1) \mid a, b \in N \cap N'$, there exists a path, with length larger than 1, from a to b, and no intermediate node on the path is in $N \cap N'\}$, $C = C_0 \cup C_1$. For every $(a, b, c) \in C$, if $c = 0$, then for any $e \in E$ with $f(e) = (a, b)$, include e in the set of *relationship-ids*; else if $c = 1$, generate a new unique *relationship-id* for (a, b) and include it in the set of *relationship-ids*.

Operator 4.4.5 (filtering by record-id): The *filtering-by-record-id* operator takes a structure graph (N, E, f) and a set N' of *record-ids* as input, and produces a structure graph with $N \cap N'$ as the set of *record-ids*, and $\{e \mid e \in E,$ and $f(e) = (a, b),$ and $a, b \in N \cap N'\}$ as the set of *relationship-ids*.

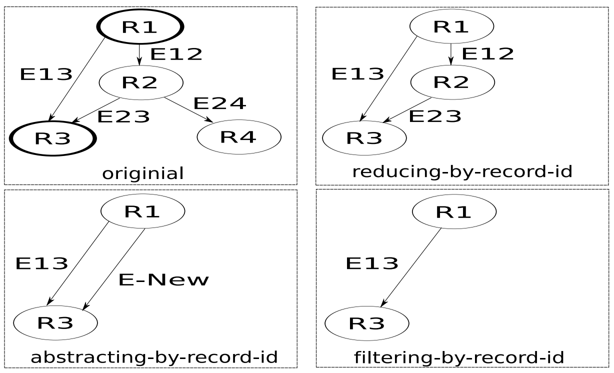


Figure 3: Constricting Structure Graph By *record-id*

Figure 3 contrasts the three aforementioned operators by applying them to the same structure graph (with {R1, R3} being the constricting set of *record-ids*). As an example, note that we can use the *abstracting-by-record-id* operator to make an "activity" view of the provenance structure.

Operator 4.4.6 (filtering by relationship-id):
The *filtering-by-relationship-id* operator takes a structure graph (N, E, f) and a set E' of *relationship-ids* as input, and produces a structure graph with $E \cap E'$ as the set of *relationship-ids* and $\{n \mid \exists e \in E \cap E',$ and $f(e) = (n, *)$ or $(*, n)\}$ as the set of *record-ids*.

Operator 4.4.7 (union):
The *union* of two structure graphs (N, E, f) and (N', E', f') is $(N \cup N', E \cup E', f \cup f')$. It is easy to verify that the result of union is still a structure graph. Similarly, we can define an intersection operator.

Operator 4.4.8 (descendant):
The *descendant* operator takes a structure graph (N, E, f) and a set N' of *record-ids* as input, and produces a structure graph with $(N \cap N') \cup \{n \mid n \in N, \exists s \in N \cap N'$, and there is a path from s to $n\}$ as the set of *record-ids*, and $\{e \mid e \in E$, $\exists s \in N \cap N'$ and e is on a path starting from $s\}$ as the set of *relationship-ids.*

Operator 4.4.9 (ancestor):
The *ancestor* operator takes a structure graph (N, E, f) and a set N' of *record-ids* as input, and produces a structure graph with $(N \cap N') \cup \{n \mid n \in N, \exists d \in N \cap N'$, and there is a path from n to $d\}$ as the set of *record-ids*, and $\{e \mid e \in E$, $\exists d \in N \cap N'$ and e is on a path ending at $d\}$ as the set of *relationship-ids.*

A Pattern Matching Operator

Sometimes, users may want to retrieve complex relationships that match a pattern.

Defintion 4.4.2: A *labeling function* is $l : NSpace \rightarrow NSpace$. Intuitively, a labeling function assigns labels to the "nodes" and "edges" of a structure graph.
Definition 4.4.3: The *label of a path* is the sequential concatenation of each node label and edge label along the path. A *path pattern* is a *regular expression* over the labels. There can be other interesting ways of defining "label of path" and "path pattern".

Operator 4.4.10 (path matching): The *path-matching* operator takes a structure graph G, a labeling function l and a path pattern PAT as input, and produces a structure graph G' defined as follows: Let P=$\{p \mid p$ is a path of G, and $l(p)$ matches $PAT\}$.

$$G' = \bigcup_{p \in P} p.$$

Figure 4 is an example of path pattern matching. And the pattern is SX^*CX^*D.

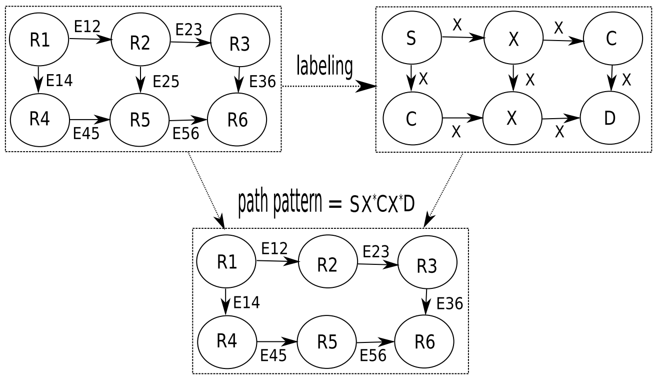


Figure 4: Path Pattern Matching

Ad Hoc Operators

Based on the concept of structure graph, it is also possible to define ad hoc operators that fit particular needs for *analysis* of provenance structure. For example, *a*) operator that returns "shortest path" between two provenance entities. *b*) operator that returns the provenance entities that have "out-degree" larger than a number.

Definition 4.4.4: The concept of *structure graph* and its relevant query operators constitute the *structure based query model* of provenance.

4.5 Combining Two Sub-models of Query

Figure 5 roughly illustrates how the content based model and the structure based model fit together to carry out provenance query.

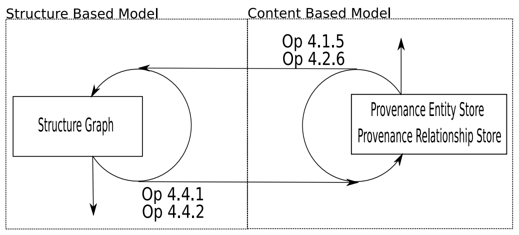


Figure 5: Query Model of Provenance

A Fictitious Example

We conclude this section with an illustrative scenario of food safety tracking. In today's globalized economy, the production and consumption of goods are often distributed at different places. For example, a manufacturer may use parts and material from different countries, and the final products are shipped around the globe. In this case, it is helpful to keep track of the provenance of goods for quality assurance, dispute settlement, etc.

Figure 6 shows an assumptive workflow of powdered milk production and consumption. Powdered milk is produced in country Z, and is exported to country X and country Y. Some confectionary companies in country X use imported powdered milk from country Z in their production of chocolate candy. In this scenario, we don't consider how data is collected, but focus on the intuition of how provenance queries are used.

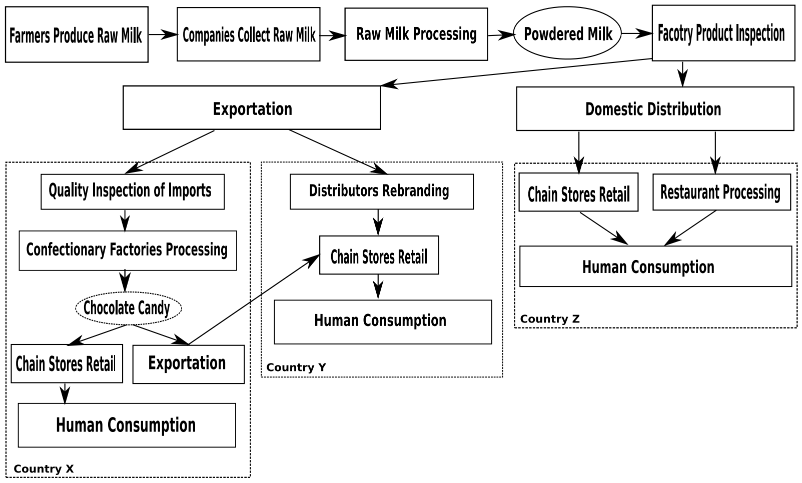


Figure 6: Powdered Milk Production And Consumption

Q1: Imagine one day, a brand of powdered milk produced in country Z is found contaminated. What brands of chocolate candy are affected in country X? How much is the total worth of the affected products?

S1: *a*) Find the problematic batch of powdered milk by content based query (by "brand name", "address" and "production time"). *b*) Get the descendants of the problematic batch by structure based query (the *descendant* operator). *c*) Conduct content based query on the descendants (e.g., "type"="chocolate candy", "address"="country X") to get affected brands of chocolate candy in country X. *d*) Use operator 4.1.6 (aggregation) to calculate the total worth.

Q2: How was the problematic batch of powdered milk produced, transported and processed to make an affected brand of chocolate candy? Was the powdered milk inspected both before and after exportation?

S2: *a*) Find the problematic batch of powdered milk and the affected brand of chocolate candy by content based queries. *b*) Relate the problematic batch of powdered milk and the affected brand of chocolate candy by using operator 4.4.3 (reducing by record-id). *c*) In the structure graph obtained from the previous step, use operator 4.4.10 to check whether there exists a path matching pattern (inspection)X^*(exportation)X^*(inspection).

Q3: There are two separate brands of chocolate candy that have received complaints. Do they share some kind of similarity in their production processes?

S3: *a*) Find these two brands by content based queries. *b*) Find the ancestors of each brand separately (operator 4.4.9). *c*) Compare the results obtained from the previous step by, e.g., intersection of structure graphs.

5 System Architecture

Figure 7 shows the architecture of `Butterfly`.

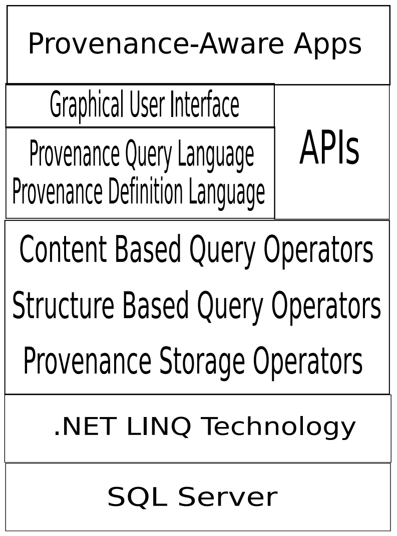

Provenance-Aware Apps
Graphical User Interface
Provenance Query Language
Provenance Definition Language
APIs
Content Based Query Operators
Structure Based Query Operators
Provenance Storage Operators
.NET LINQ Technology
SQL Server

Figure 7: `Butterfly` System Architecture Stack

The implementation is in the initial stage. We use Microsoft Language Integrated Query (LINQ) and SQL Server as the underlying storage engine. Upon that, we are building a set of operators for creating and manipulating provenance data. Currently, we have not focused on operator optimization issues.

6 Related Work

Data provenance, also called data lineage, describes how a piece of data was obtained from its predecessor. Many projects in data provenance rose from the domain of scientific computing and experiments. Survey paper [14] develops taxonomy of provenance techniques to compare nine key provenance systems. Survey paper [5] proposes a meta-model for architectures of lineage retrieval systems. The Chimera project [7] proposes the idea of explicit representation of computational procedures and their invocations in a virtual data catalog in the Data Grid environment. In the myGrid project [13], semantically rich execution logs [16] are automatically produced during work-flow invocations. CMCS [12] develops a meta-data infrastructure for chemical science where lineage relationships between data entries can be visualized. Paper [4] proposes a lineage meta-data model and associates meta-data to every constituent of a work-flow. In database research area [6, 2, 3], the provenance problem focuses on locating the source data items that produced or influenced the production of the result data items, rather than the process of transformations that was applied to obtain the result. Paper [10] names such type of provenance as input provenance. Paper [2] and [3] use logging to store provenance information during execution of simple queries. More particularly, paper [2] logs the causal relationships between output rows and source rows. Paper [3] tags every piece of data at attribute level in the source tables with a unique identifier, propagates the identifiers along with the data they tagged during the query processing. There are other ad hoc systems that apply the idea of storing and utilizing provenance information for a variety of application specific needs [9, 15, 1]. Project PASOA [8] attempts to provide a general provenance architecture that satisfies the need of applications whose system architectures are service oriented.

7 Conclusion And Future Work

In this paper, we introduce `Butterfly`, a provenance management system. The advantages of `Butterfly` are *a*) it is agnostic of domains. *b*) its semantic model is flexible, and its algebraic query model is extensible and intuitive. These make `Butterfly` a strong candidate for a general purpose management system for provenance.

Future work can be categorized as follows:

- Richer semantic model elements of provenance
- Query operators optimization
- Provenance storage optimization
- User interface design and system implementation
- Novel provenance-driven applications

References

[1] R. Becker, J. Chambers, *Auditing of Data Analyses.* in SIAM Journal on Scientific and Statistical Computing, 1988.

[2] O. Benjelloun, A. D. Sarma, C. Hayworth, J. Widom, *An Introduction to ULDBs and the Trio System.* in IEEE Data Engineering Bulletin, Special Issue on Probabilistic Databases, volume 29, 2006.

[3] D. Bhagwat, L. Chiticariu, W. C. Tan, G. Vijayvargiya, *An annotation management system for relational databases.* in Proc. of the Intl. Conf. on Very Large Data Bases (VLDB), 900–911, 2004.

[4] R. Bose, J. Frew, *Composing lineage metadata with XML for custom satellite-derived data products.* in Proceedings of 16th International Conference on Scientific and Statistical Database Management, 2004.

[5] R. Bose, J. Frew, *Lineage retrieval for scientific data processing: a survey.* in ACM Comput. Surv., volume 37:1–28 2005.

[6] Y. Cui, J. Widom, J. L. Wiener, *Tracing the lineage of view data in a warehousing environment.* in ACM Transactions on Database Systems (TODS) 25 (2000) 179–227, 2000.

[7] I. Foster, J. Vockler, M. Wilde, Y. Zhao, *Chimera: A virtual data system for representing, querying, and automating data derivation.* in Fourteenth International Conference on Scientific and Statistical Database management, Proceedings, 2002.

[8] P. Groth, S. Jiang, S. Miles, S. Munroe, V. Tan, S. Tsasakou, L. Moreau, *An Architecture for Provenance Systems.* Technical report, Electronics and Computer Science, University of Southampton, 2006.

[9] D. P. Lanter, *Design of a Lineage-Based Meta-Data Base for GIS.* in Cartography and Geographic Information Systems, vol. 18, 1991.

[10] S. Miles, P. Groth, M. Branco, L. Moreau, *The Requirements of Using Provenance in e-Science Experiments.* in Journal of Grid Computing, Springer, 2007.

[11] L. Moreau, J. Freire, J. Futrelle, R. McGrath, J. Myers, P. Paulson, *The Open Provenance Model.* Technical Report 14979, ECS EPrints repository, 2007.

[12] C. Pancerella et al, *Metadata in the Collaboratory for Multi-Scale Chemical Science.* in 2003 Dublin Core Conference: Supporting Communities of Discourse and Practice-Metadata Research and Applications, 2003.

[13] R. D. Robert, D. Stevens, A. J. Robinson, C. A. Goble, *mygrid: Personalised bioinformatics on the information grid.* in Bioinformatics, 19 Suppl 1:302–304, 2003.

[14] Y. L. Simmhan, B. Plale, D. Gannon, *A Survey of Data Provenance Techniques*. Technical Report IUB-CS-TR618, Indiana University,Bloomington, 2005.

[15] A. Vahdat, T. Anderson, *Transparent Result Caching*. in the Proceedings of the USENIX Annual Technical Conference (NO 98), 1998.

[16] J. Zhao, C. Goble, M. Greenwood, C. Wroe, R. Stevens, *Annotating, linking and browsing provenance logs for e-science*. in Proc. of the Workshop on Semantic Web Technologies for Searching and Retrieving Scientific Data, 158–176, 2003.