THE MITRE CORPORATION

Provenance Capture and Use: A Practical Guide

M. David Allen, Len Seligman, Barbara Blaustein, Adriane Chapman

6/15/2010

MITRE Product # MP100128

Approved for Unlimited Public Release, Case 10-2146

ABSTRACT. There is a widespread recognition across MITRE's sponsors of the importance of capturing the provenance of information (sometimes called lineage or pedigree). However, the technology for supporting capture and usage of provenance is relatively immature. While there has been much research, few commercial capabilities exist. In addition, there is neither a commonly understood concept of operations nor established best practices for how to capture and use provenance information in a consistent and principled way. This document captures lessons learned from the IM-PLUS project, which is prototyping a provenance capability that synthesizes prior research; the project is also applying the prototype to government application scenarios spanning defense, homeland security, and bio-surveillance. We describe desirable features of a provenance capability and trade-offs among alternate provenance approaches. The target audience is systems engineers and information architects advising our sponsors on how to improve their information management.

Contents

1 Introduction

1.1 Background and Purpose of this Document

There is a widespread recognition across MITRE's sponsors of the importance of capturing the *provenance* of information (sometimes called lineage or pedigree). However, the technology for supporting capture and use of provenance information is relatively immature. While there has been much research, there are few commercial capabilities. In addition, there is no commonly understood concept of operations or best practices for how to capture and use provenance information in a consistent and principled way. In addition, some government organizations are now creating provenance schemas, including PREMIS, from the Library of Congress, and Upside, for Navy Joint Air Missile Defense; these describe the provenance information to be managed, but do not address how this information is to be captured, stored or queried. So, *while provenance is widely considered important, our sponsors lack guidelines for how to develop and incrementally improve provenance collection and use.*

Since October 2007, the IM-PLUS[1] project at MITRE has been prototyping a provenance capability that synthesizes prior research. In addition, we have applied the prototype to government application scenarios spanning defense, homeland security, and bio-surveillance. From these experiences, we have learned many lessons about desirable features for a provenance capability and important trade-offs.

The purpose of this document is to capture the IM-PLUS lessons learned so that they can be applied to address our sponsors' needs for data provenance. The target audience is systems engineers and information architects advising our sponsors on how to improve their information management.

This is a living document, which is continually evolving as we learn additional lessons. We welcome feedback on how to make it more useful.

1.2 What is Provenance?

There is no standard, universally accepted definition of provenance nor of the closely related terms lineage, pedigree, and context. In fact, each of MITRE's sponsors seems to use a different word, and the definitions often conflict.

In this report, we use the term "provenance" because it is the most prevalent one in the research community.[2] We adopt the definition of Harvard's provenance group, which is fairly typical among researchers:

[1] IM-PLUS stands for "Information Management with Privacy, Lineage, Uncertainty, and Security". While the original proposal focused on interactions of all four of the PLUS properties, our sponsor requirements led us to focus mostly on lineage, with some consideration of interactions with security and privacy.

[2] As evidence of this, note the existence of international workshops on Theory and Practice of Provenance Systems (TaPP), the International Provenance and Annotation Workshop (IPAW), and a proposed standard model among researchers called the Open Provenance Model (OPM).

Data provenance is "information that helps determine the derivation history of a data product...[It includes] the ancestral data product(s) from which this data product evolved, and the process of transformation of these ancestral data product(s)."[3] Essentially, provenance provides a data "family tree," helping consumers of data to interpret it properly and determine how much they should trust it. We use "lineage" interchangeably with "provenance."[4]

Since there is often considerable confusion over related terms, we propose the following definitions, and will use them in the remainder of this report:

- *Context* is domain-relevant information about the state of the world at the time information is created or modified. Some may be collected along with the provenance information (e.g., water temperature at time of buoy reading), while other context may already be available as separately maintained data (e.g., the tolerances of sensors).
- *Data quality assessments* are estimates of confidence, accuracy, and/or timeliness of information. These assessments may be automatically or manually generated**.**
- *Data pedigree* is the combination of the data's provenance, context, and data quality assessments.

This report is about provenance, but also discusses how to attach annotations to the data family tree. These annotations provide needed data structures for both context and data quality assessments. Thus, a well designed provenance model provides the necessary infrastructure on top of which a broader pedigree capability can be layered.

1.3 Why is Provenance Important?

MITRE's customers need provenance to enable the following:

- *Consumer understanding and trust in net-centric environments*. When users can access vast quantities of information from beyond their traditional stovepipes, they need to understand where that information comes from so that they can make a decision on whether or not to trust it for their purpose. One such use of provenance is to help determine if two reports of an event are actually derived from the same original report or if they are truly independent and therefore represent corroborating evidence.
- *Understanding of the consequences of a data modification cyber-attack.* A key issue for mission assurance in the face of such an attack is data "taint analysis"—i.e., understanding downstream effects of suspect or corrupted data. In addition, the provenance family tree can be used to trace backwards to see if upstream processes (e.g., a sensor fusion algorithm) may have been compromised and have produced the bad data.
- *Information Discovery*. Provenance can improve both the accuracy (i.e., "precision") and completeness (i.e., "recall") of information discovery. First, it provides a useful way for consumers to filter out unwanted information, by only returning certain sources or excluding ones derived from some known bad source, etc. Second, it provides a new

[3] K.-K. Muniswamy-Reddy, D. A. Holland, U. Braun, and M. I. Seltzer, "Provenance-Aware Storage Systems," *USENIX Annual Technical Conference*, pp. 43-56, 2006.

[4] "Lineage" has been used most often in work that traces provenance for information that solely resides in relational databases. Stanford's Trio project (http://infolab.stanford.edu/trio/) typifies that work.

mechanism for discovering useful sources that have not previously been considered. For example, an analyst could trace backwards in the provenance to discover which sources were used to produce some favorite product and subsequently see which other products used those same sources. In recognition of its importance for discovery, the Defense Discovery Metadata Standard (DDMS)[5] contains fields that describe data pedigree.

The need for provenance information is explicitly called out in the DoD's Net-Centric Data Strategy (NCDS)[6] as a part of the need to "make data trusted". The NCDS envisions a future state where "*users and applications can determine and assess the authority of the source because the pedigree, security level, and access control level of each data asset is known and available*"[7]. This high-level goal is then given a specific action that all in the DoD must take to help accomplish the goal:

> **Associate Data Pedigree and Security Metadata**[8]
> The Resource Descriptors elements of the DDMS (…) allow identification of the author, publisher, and sources contributing to the data, allowing users and applications to assess the derivation of the data (i.e., data pedigree). This metadata allows users and applications to select data from known sources. Reliable and quality sources will become more widely used, enhancing overall data quality throughout the Enterprise as more data sources become visible.

Appendix B presents additional examples of government data strategy documents that recognize the importance of provenance.

1.4 Basic Provenance Concepts

While there is no standard for provenance, there are certain elements that appear repeatedly in many provenance capabilities. For example, most have these basic concepts:

- *Actors*: these are the things (e.g., humans, organizations, sensors, computer programs that spawn other programs) that initiate processes or that can own the objects described in provenance;
- *Processes*: these are the activities that manipulate data. These can be machine executable (e.g., scripts, programs) or can be the result of human interaction.
- *Data*: these are collections of symbols that represent statements about the world. Note that provenance systems are not interested in the contents of the data (i.e., the *base data*), since this is the responsibility of applications. Instead, provenance systems just maintain information *about* the data, such as where it came from and (optionally) provide a pointer to the base data.

The fundamental data structure for provenance is a directed acyclic graph (DAG), where the nodes are process invocations and information about data, called process and data nodes respectively. These are linked together by edges that describe the dependencies. Provenance is a

[5] http://metadata.dod.mil/mdr/irs/DDMS/
[6] http://cio-nii.defense.gov/docs/Net-Centric-Data-Strategy-2003-05-092.pdf
[7] Net-Centric Data Strategy Goal 3.5: Enable Data to be Trusted
[8] Net-Centric Data Strategy, section 3.5.1

DAG and not an acyclic graph because of the temporal aspect of provenance. Every node in a provenance graph represents a particular data or process invocation at an instant in time.

Ideally, invocations and data would strictly alternate in a provenance graph; the alternating boxes (process invocations) and ovals (data) of such a graph resemble a *beaded necklace*. However, since it is often not possible to capture every single invocation and intervening data item, not all provenance graphs are required to be beaded necklaces. Many may contain sequences of nothing but reported invocations, without intervening data; similarly, data nodes can be directly connected to their descendents without intervening process nodes. The contents of the graph are a consequence of how the provenance was gathered or recorded, which will be discussed in depth in later sections.

1.5 Example

Figure 1 shows an example provenance graph, in which data from a variety of maritime sensors are acted upon by a number of processes and ultimately produce a track database. Following the conventions of the provenance research community, ovals represent data nodes, while rectangles represent process invocations. Where a data node feeds a process node, this means that the data was an input to that process.

To illustrate, "Geo-temporal Normalization and Fusion Algorithm 1" in Figure 1 has three sensor inputs—Processed Acoustic Sensor Data, Sea State Sensor Data, and Sonar Sensor Data—and two other inputs: Sensor Characteristics (i.e., technical characteristics of the different sensors) and Sea Object Signatures (i.e., patterns that might help identify different kinds of objects, for example a whale vs. a submarine). That process in turn produces one output: the Maritime Object Database.

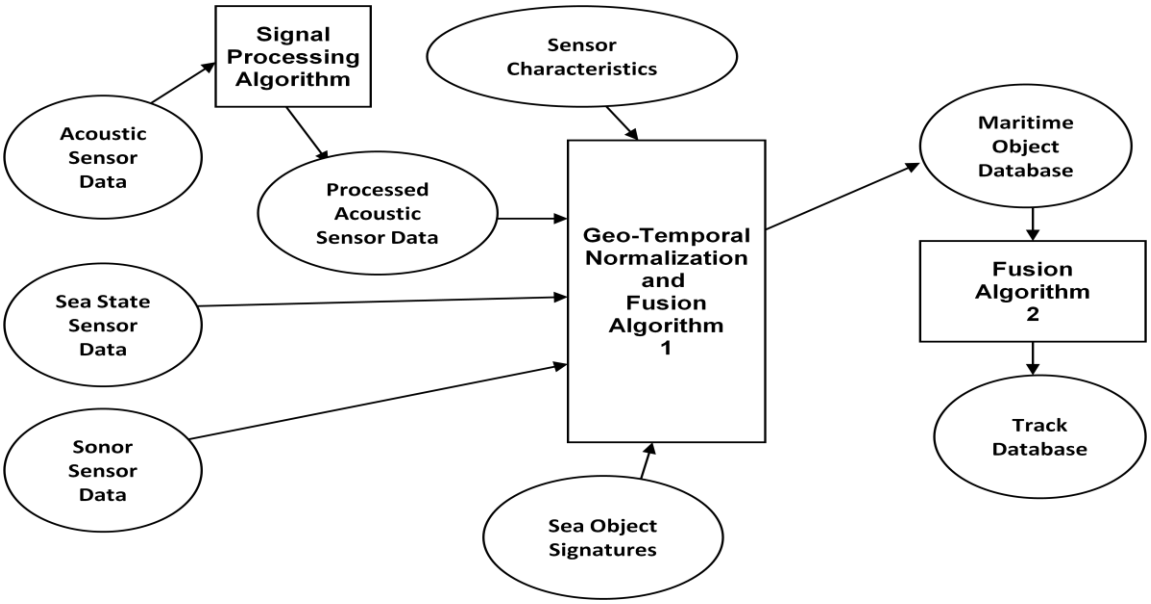

Figure 1. Example Provenance Graph for Producing a Track Database

It is useful to traverse a provenance graph in both the forward and backward directions. For example, one must do a forward traversal to answer the question: "What downstream data was affected by Sonar Sensor Data?" One must traverse the provenance graph backwards to answer the question: "What processes contributed either directly or indirectly to the creation of the Track Database?" Because forward and backward provenance queries are so common, we sometimes use shorthand names: "fling" (i.e., forwards lineage) and "bling" (i.e., backwards lineage).

1.6 State of the Practice

The current state of the practice is *manual population* (also called manual tagging)—that is, some human must manually enter one or more data fields that describe the provenance. Typical field names include "source", "pedigree", "lineage", or "provenance." For example, the Defense Discovery Metadata Specification (DDMS) includes a field for the "source" of a data asset.

There are many limitations of the manual population approach.

- Because it relies on a manual process, the provenance is often left blank.
- Even when it is filled in, there is no mechanism to ensure that it is done consistently—i.e., is it the original source or the immediate ancestor?
- When it is filled in, it is "one-hop" at best; indicating an immediate ancestor, with no linkage further back; no information is available about what intervening processes (e.g., algorithms, human review) acted upon the data.
- Because the field is often a free-form text field (as it is in DDMS), there is no way to capture data's "family tree."
- Manual population makes "fling" queries extremely difficult since finding the "fling" of a particular object requires searching the universe of data that might have mentioned that object as its source.

For provenance to be widely captured and maximally useful there must be mechanisms to automatically capture it and to populate information's full family tree.

A number of research efforts have explored automatic capture of provenance. Major threads include: capture of scientific workflows,[9,10,11,12] derivation of data in relational databases,[13,14] and provenance-aware storage systems.[15] There are few commercial products with capabilities

[9] Callahan, S.P., J. Freire, E. Santos, C.E. Scheidegger, and C.T.S.H.T. Vo, "VisTrails: Visualization meets Data Management," *SIGMOD*, 2006.
[10] Davidson, S., S. Cohen-Boulakia, A. Eyal, B. Ludascher, T. McPhillips, S. Bowers, and J. Freire, "Provenance in Scientific Workflow Systems," *IEEE Data Engineering Bulletin*, vol. 32, no. 4, 2007.
[11] Moreau, L., B. Ludäscher, I. Altintas, R.S. Barga, S. Bowers, S. Callahan, et al., "Special Issue: The First Provenance Challenge," *Concurrency and Computation: Practice and Experience*, vol. 20, 2008.
[12] Oinn, T., M. Greenwood, M. Addis, M.N. Alpdemir, J. Ferris, K. Glover, et al., "Taverna: lessons in creating a workflow environment for the life sciences: Research Articles," *Concurr. Comput. : Pract. Exper.*, 18(10), 2006.
[13] Benjelloun, O., A.D. Sarma, C. Hayworth, and J. Widom, "An Introduction to ULDBs and the Trio System," *IEEE Data Eng. Bull.*, vol. 29, no. 1, 2006.
[14] Buneman, P., J. Cheney, and S. Vansummeren, "On the Expressiveness of Implicit Provenance in Query and Update Languages.," *ICDT*, 2007.
[15] Munroe, S., S. Miles, L. Moreau, and J. Vázquez-Salceda, "PrIMe: a software engineering methodology for developing provenance-aware applications," *SEM*, 2006.

for automatically collecting provenance[16]; we are currently aware of none that offer general purpose, enterprise provenance management.

2 An Abstract View of a Provenance Capability

Figure 2 presents an abstract, implementation-independent view of a provenance capability. This view will help clarify aspects of our proposed data model for provenance (Section 3). In addition, we will refer to this view when we consider alternative provenance deployment strategies (Section 4).

While the implementation details can vary considerably, we believe that any practical provenance manager must support at least the following interfaces:

- *Report*. This is an interface through which various actors can report provenance. The provenance could be self-reported by applications or could be captured by software modules that observe applications and/or human actors. We call such software modules "capture agents." Once the basic provenance information is reported, it cannot be changed. This is because it represents a historical record of actual executing processes. The fact that process p_1 executed at time t_2 and produced output data d_3 does not change, even if d_3 is later modified. We discuss these issues in greater detail in Section 4 on Provenance Capture.
- *Annotate*. To support different kinds of analysis, actors may need to attach different kinds of metadata to provenance information. Examples include confidence and arbitrary application specific context (e.g., the operating system version, or the POC for a particular object). Unlike basic provenance information captured through the "report" interface, annotations can change. For example, analyst Joe may want to indicate that a particular data item should not be trusted. Joe may want to later revise that assessment. Also, analyst Sue could attach a different confidence to the same data item as long as she had the privileges to do so.
- *Retrieve*. Once provenance information is captured, users and applications need the ability to retrieve it to support their analyses. We discuss options for supporting retrieval in Section 4.6.
- *Administer*. Administrators for the provenance system need an interface for administering security and performing other administrative functions. A key issue is distributing authority, since it is not scalable (at least for an enterprise capability) to have a single administrator determine privileges for all provenance information. We address these issues in a 2009 workshop paper[17] and do not discuss them further in the current document.

[16] Currently, we are aware only of the Pedigree Management Assessment Framework (http://www.raesoftware.com/solutions.htm), which maintains provenance of Microsoft Office documents when edits are done with the tool's plug-ins.

[17] Rosenthal, A., L. Seligman, A. Chapman, and B. Blaustein, "Scalable Access Controls for Lineage," *First Workshop on Theory and Practice of Provenance Systems (TaPP)*, 2009.

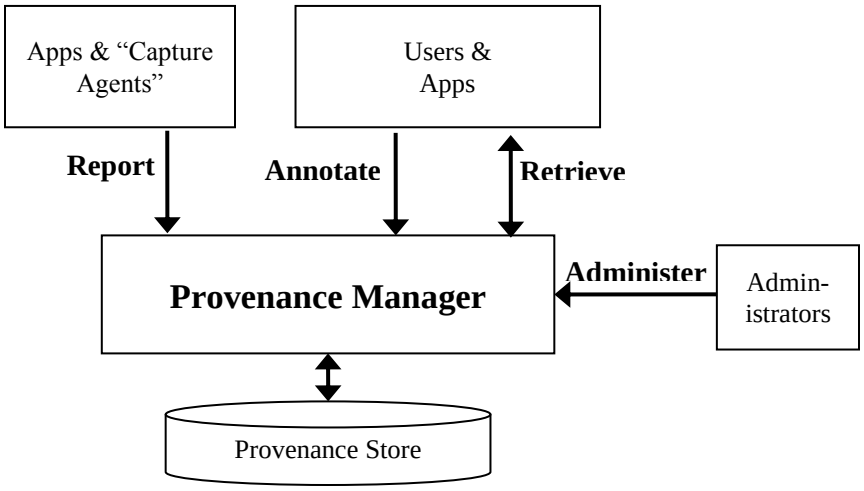

Figure 2 An abstract view of a provenance capability

In this discussion, we abstract away implementation details of the underlying data store for provenance information. Nevertheless, the choice of underlying provenance store can affect the capabilities that can be offered. For example, it would not be practical for the Retrieve capability to support complex queries unless that was already supported by the underlying provenance store.[18]

3 Provenance Model

The foundation of any approach to provenance is development of a suitable model for representing provenance information. The IM-PLUS project began by conducting a survey of the state of the art in provenance representation and exploitation and then selected those features that best met the requirements of MITRE's customers.

The state of the art can be summarized in terms of two different approaches to the problem. The first approach focuses on provenance within relational databases, and in particular how to track the derivation of particular records within a database. For example, Stanford's Trio project[19] tracks the query language operations (such as inserts, modifications, and creation of views) that explain the presence of particular records in a derived dataset. Like most database-oriented approaches, those operations are limited to those of SQL, the standard query language for relational databases. The second approach focuses on the study of workflows consisting of various processing steps, which can consist of arbitrary computations, with few or no assumptions placed on the kinds of data which result from those processing steps. An example of such a workflow might be a scientific experiment that gathers data from one source, stores it in a

[18] An example of such a complex query: "tell me all invocations of fusion algorithms created by the Defense Intelligence Agency which in some way contributed to the creation of data products X or Y."
[19] O. Benjelloun, A. Das Sarma, A. Halevy, M. Theobald, and J. Widom. Databases with Uncertainty and Lineage. *VLDB Journal*, 17(2):243-264, March 2008.

database, renders an image using the data, and performs analysis on that image. IM-PLUS ultimately selected the "workflow" approach to provenance, as it more closely resembles some of the distributed applications that the government needs to manage. Additionally, the "relational only" approach was rejected since few of our customers' important workflows can be expressed solely in SQL.

Our design was influenced by the Open Provenance Model (OPM)[20], an effort by several provenance researchers to synthesize the models of various workflow-based approaches. In addition, we focused on the goals of extensibility and generality. The provenance model must be general enough to capture the majority of reasonable provenance use cases. Further, it must be extensible because no single model can sufficiently capture all requirements; the ability to augment the model to address local domain-specific needs is highly desirable. Based on OPM and the review of the state of the art, IM-PLUS designed the provenance model that will be described in this paper.

We refined our model based on a number of real scenarios used by the government, to test that the model was sufficiently general to be able to represent many different domains, and also extensible enough to capture domain-specific requirements. Those scenarios in turn informed improvements to the model.

Note that in this work, we restrict ourselves to an abstract provenance model. This model is consistent with all emerging government standards for provenance, and allows us to highlight the needs of an overall provenance system without getting mired in the domain specific details of all current standards for provenance.

3.1 Basics

As noted above, the fundamental data structure for provenance is a graph, where the nodes are process invocations and information about data, called process and data nodes respectively. We provide a detailed description of the information IM-PLUS maintains about nodes in Section 3.4.

There are four types of edges in a provenance graph, where the type depends entirely upon the types of the nodes that the edge connects:

- *Generated*, which connects an invocation to the data it outputs
- *Input to*, which connects data to an invocation that it feeds
- *Contributed*, which connects a data item to another data item, indicating that the contents of the former contributed to the latter
- *Triggered*, which connects an invocation to another invocation, indicating that the former caused the invocation of the latter without an intervening data input

3.2 Invocations, Processes, and Workflows

Most provenance models distinguish between processes and invocations. Processes are abstract definitions of tasks to be performed. For example, converting a document from text to a PDF is

[20] L. Moreau, J. Freire, J. Futrelle, R. McGrath, J. Myers, and P. Paulson, *The Open Provenance Model*, University of Southampton, 2007.

a process that can be applied to data. But when that process is actually invoked with a particular text file, provenance information is created. The "invocation" of the "convert" process is then captured in a provenance graph, along with its input (the text document) and its output (the PDF document). It is important to note that processes and invocations are distinct. As an example, the addition process, applied many thousands of times to different inputs, produces many thousands of distinct invocations.

Effective provenance capture requires capturing information on invocations, and not just data. Invocations describe how and why the data changed. Provenance graphs that contain data only can certainly indicate which content influenced which other content, but without invocation information, there is no explanation of how that influence occurred.

In addition to capturing information on invocations, some provenance systems also capture knowledge about the processes of which invocations are instances. For example, consumers of provenance information may want to know the version, author, release date, licensing information, etc., for a particular piece of software. While this could be replicated for each invocation, this is in general a bad practice that violates the principles of data normalization.

Just as we draw the distinction between "process" and "invocation", it is important to draw a distinction between an abstract workflow specification, and an actual provenance graph. An abstract workflow may be specified as a pre-determined set of steps. Execute A, then B, then C. The focus of a provenance model is to capture information about what actually took place. As such, the model does not capture abstract workflow specifications because they are not part of the record of "what happened". Abstract workflows do though typically contain things such as loop constructs, conditional execution statements and other artifacts. These additional artifacts can be quite useful depending on the application scenario; application developers may choose to capture and manage these separately, but outside of the scope of the provenance application.

3.3 Principles and Scope of the Provenance Model

This section describes guiding principles behind the development of the IM-PLUS provenance model.

Principle 1: Provenance graphs are rarely complete

No provenance graph should be assumed to be complete, as it is only a record of what occurred through a particular point in time. Provenance graphs may grow due to subsequent activity of merging of additional actors' information, and the information they contain may itself be incomplete. As long as any participant in the graph continues to exist, the graph may evolve over time to add new members. In addition, some processing steps may never be reported to the provenance service. As a result, one should not assume that any provenance graph is complete.

Principle 2: Some provenance is better than no provenance (partial value assumption)

It is not necessary to capture a full or complete provenance graph in order for users to derive operational benefit from that graph. Additionally, it is not necessary for a graph to be fully linked in order for it to be useful. Often situations arise where key linkages cannot be captured for various technical or operational reasons. For example, application A calls B, which calls C, which calls D. Ideally, this would form the chain A -> B -> C -> D. If it is not possible to log

the interaction between B and C, the result is actually two distinct provenance graphs; one indicating that A -> B, and the second indicating C -> D. Provenance, even if incomplete, provides useful context information that aids users' trust decisions, especially when compared to current practice in which provenance is rarely captured at all.

Principle 3: Management of base data is the responsibility of underlying systems and is out of scope for a provenance capability.

The base or underlying data (the contents of the invocation, process, or data item being described) are not within the scope of a provenance capability, and are best left to underlying systems to avoid duplication. Provenance capabilities do not manage or record base data for many different reasons; first, the base data has different sensitivities than the provenance records, which are often best addressed by the source system. Second, base data can be quite large and unwieldy; duplicating it and managing it separately as part of a provenance system is often technically unworkable. Third, provenance graphs track changes in data that are sometimes quite rapid. Imagine a workflow that repetitively edited a very large database. Managing the base data in such scenarios amounts to implementing a historical database to store all past versions of the data, which is itself a challenging research issue.[21] It would be a mistake to take on this challenge in early efforts to field provenance capabilities.

Principle 4: Provenance capture/display/processing must happen with minimal or zero change to legacy systems.

Given resource limitations and unanticipated information flows, it is essential to minimize the changes required to legacy systems to enable provenance capture. While a small number of applications may derive sufficient value from provenance to justify such modifications, most will need to rely on a more generic and scalable capture strategy. As such, this paper focuses on approaches that allow existing applications to continue to run as-is. At the same time, it is important that a provenance service provide a clean API so that future applications being built from the ground up have the ability to report provenance at any granularity that is appropriate to the specific application.

Principle 5: Provenance management is general purpose infrastructure useful to many applications.

Frequently, potential users of provenance have a particular use case in mind. For example, an analyst might want to track data products derived from a particular algorithm, and then mark them as "merged" to indicate that they are basically the same thing, and that the organization need not continue to support duplicate effort to generate them separately. Such analyses are a key benefit of a provenance capability. But the focus of the provenance capability is to enable such analyses, not to support any specific set of analysis use cases. This orientation allows the capability to provide a generally reusable service that can be tailored to a variety of application needs.

3.4 Core and Extension Provenance Model

As noted in Section 3.1, provenance information is captured as a graph, where the nodes are either data items or process invocations and the edges show dependencies among the nodes. As

[21] Sarda, N.L., "Extensions to SQL for Historical Databases," *IEEE Transactions on Knowledge and Data Engineering*, vol. 2, no. 2, 1990.

described in Principle 3 of the previous section, a provenance service does not manage base data. Thus, a data node for Database-3 can optionally contain a handle (e.g., a URL) for the underlying data source, but the provenance service is not responsible for actually providing access to that source.

Based on the abstract model described in Section 2, we note two main kinds of provenance information:

- *Basic provenance information*. This is the immutable information on process invocations, data items, and their dependencies captured via the Report interface. Once this information is captured, it never changes, since it is a historical record of what happened.
- *Annotations*. This is additional metadata attached to objects in the basic provenance information, for example, that analyst Joe thinks that a particular data item is not trustworthy. Annotations can be added, deleted, and modified after initial provenance capture by any authorized user or application.

Having made this distinction between basic provenance information and annotations, the next step is to determine what specific information should be captured about data and process executions. *Because application needs vary so widely, we advocate the use of a "core plus extensions" design pattern for both the basic provenance information and annotations.* Such a pattern is often used when there are generic requirements that are broadly applicable and also enterprise-specific or community-specific requirements. Other examples using the "core plus extensions" approach include the Universal Core and National Information Exchange Model, which are being adopted widely in the DoD and Departments of Justice and Homeland Security respectively.

Figure 3 shows a simplified version of the IM-PLUS core provenance model. The model consists of nodes and edges that connect those nodes. There are four subtypes of node, each of which has a small number of attributes associated with it.[22] As shown, the number of properties captured about provenance objects is quite small, and is focused only on those that are almost universally useful to users of provenance, such as when the object was created, who owns it, and so on.

While nodes, edges, and the predefined subtypes of Node are fairly straightforward, a couple of items in the model need further explanation.

The "Invocation parameters" property of Invocation refers to the connection between inputs, outputs, and invocations. The object and edge model provides enough information to determine which inputs and outputs an invocation used; invocation parameters allow the user to distinguish which input object played which role, i.e. if an invocation takes two parameters X and Y, the user will need to know which input object was the X, and which was the Y.

AttributeValuePair provides a way for any authorized user to attach annotations (i.e., descriptive metadata, sometimes called "context") to nodes. Because there is a 1-to-N relationship between nodes and attribute value pairs, different users can annotate the same node with the same attribute name. For example, Joe and Fred could each attach an annotation for "confidence" but

[22] The distinctions between invocations, processes, and workflows are discussed in Section 3.2.

with different values to a data node corresponding to a single intelligence report. Attribute value pairs are infinitely extensible. If no preexisting attribute exists for something you want to describe (e.g., "POC phone num"), a user can create an attribute value pair with the desired attribute name. Though the mechanism itself is completely flexible, some organizations may want to insist that only already defined attributes can be used and that only certain individuals (or roles) can define these attributes. Others organizations may want to allow complete freedom to add attributes. Clearly, the use of defined vs. free attributes has the same issues as in classic data management problems, such as who administers the vocabulary for defined attributes vs. the wild west of free attributes; in many respects, provenance is just data, and the same engineering principles and trade-offs apply.

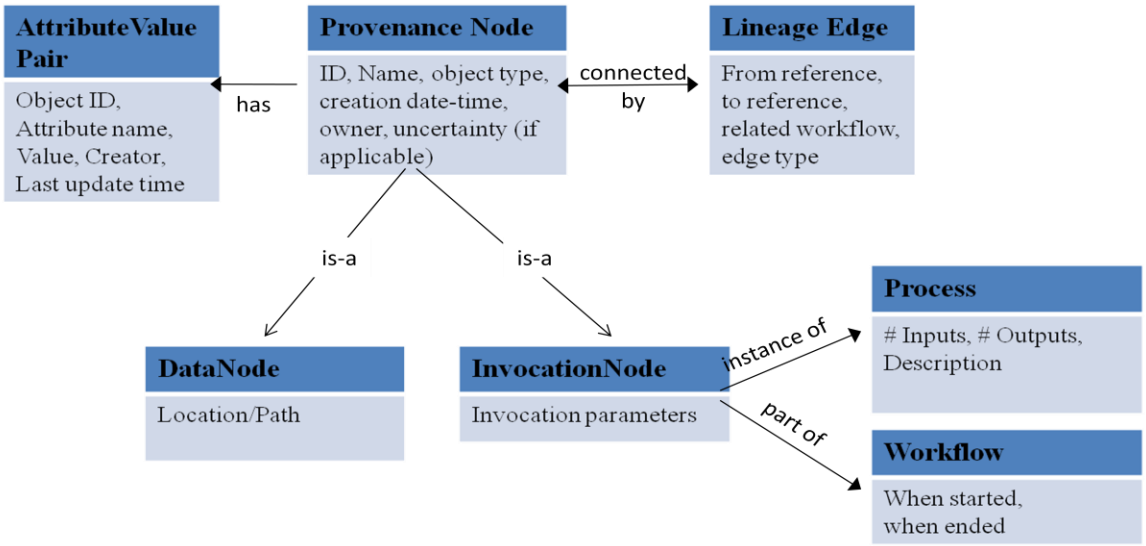

Figure 3 IM-PLUS Core Provenance Model

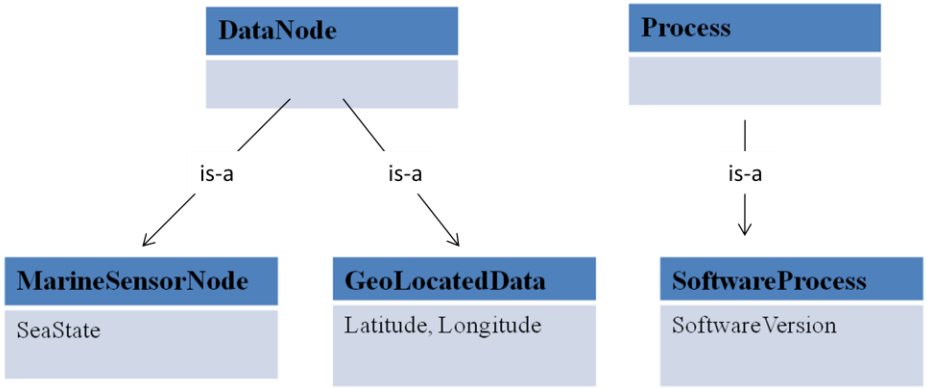

Figure 4 Example Domain-Specific Extensions

Figure 4 illustrates a different way the core provenance model can be extended, with domain-specific subtypes of data nodes and processes. In this example, two new subtypes of DataNode have been defined, one for a marine sensor node which records the "sea state" at the time of provenance capture and another for geo-located data, which here shows latitude and longitude being captured. The question comes up: wouldn't this information already be captured in the

base data, and didn't you already argue that a linage store should not normally replicate base data? By default, base data should not be captured in provenance. However, there are some situations where certain details of a message are so critical to its identity (such as the latitude and longitude of geo-located data) that capturing them is simply required for most queries. This example illustrates how this can be done. The figure also shows a domain-specific subtype of Process, called SoftwareProcess, which has a property "software version".

The model presented in the figures above is certainly not sufficient for all sponsor requirements, but it has been demonstrated to be useful as a starting point for extension. Additionally, it does capture enough information to be able to answer a broad number of useful provenance queries for users. In particular, sponsors are likely to encounter the need for sophisticated access control policies, nuanced object annotations with richer data types, and more detail on who the actors are and what their affiliations are. As new needs and provenance query requirements arise, the model can be extended and specialized to address those requirements, without requiring that each new provenance capability re-invent the overall approach.

4 Capturing Provenance: How

A provenance approach is only useful if it results in sufficient provenance capture to support needed analyses. This section presents architectural options for capturing provenance along with their strengths and weaknesses. We consider the following options:

- Observation at multi-system coordination points, including three examples
- Application reporting
- Manual capture supported by task-specific editors
- Operating system observation
- Log file parsing

These options are not mutually exclusive; complex computing environments will often necessitate the use of several approaches simultaneously.

4.1 Observation at Multi-System Coordination Points

Since it is impractical to modify most existing applications to log provenance, the least intrusive and most general place to focus is on points of coordination between multiple systems. Distributed systems typically fall into one of a number of different architectures or categories, such as client/server, or n-tiered architectures. In almost every case, the distributed system contains a point of coordination between multiple systems. In the client-server case, the point of coordination is the server. No matter how many clients connect and participate in the distributed system, they must all inevitably connect through that server. The same is true of the intermediate application in N-tier architecture, or a single coordinating process shared amongst a set of tightly coupled systems.

One of the challenges of multi-system provenance logging is the limited ability to introspect into the functioning of processes. Because multi-system provenance logging typically looks at interaction patterns between applications, it often is not focused on logging how each application

accomplishes its task. As a result, the approaches defined in this section normally treat applications as opaque "black boxes". The consequence is that as long as the externally-facing interface of an application does not change, (i.e. its required inputs and produced outputs) there may not be any way for the system to know whether or not the underlying implementation changed. From an architectural perspective, this is indeed a desirable characteristic, but it may limit the fidelity of analyses that this type of collection can enable. (For more on these issues, see Section 5.1, which discusses granularity issues.)

Despite these limitations, observation at multi-system coordination points will often be the best collection option. It is generally the cheapest option to deploy, because it eliminates the need for manual population or modification of applications.

The following sections detail various types of coordination points in distributed systems, and how provenance capture can be implemented in those contexts. While the specific examples that will be discussed may not suit all environments, the same pattern can be applied, and the same benefits realized, by finding the point of coordination within a given distributed system.

4.1.1 Enterprise Service Buses

Enterprise service buses refer to middleware based on recognized standards, which provide fundamental services for complex architectures via event-driven messaging. An enterprise service bus (ESB) is an abstraction on top of a common messaging architecture, allowing multiple applications to be coordinated by routing and sequencing messages between those applications.

Enterprise Service Buses are often confused with implementations of service oriented architectures (SOA). An ESB does not implement a SOA, but provides a number of features that can be used to implement an SOA. One of the main features of the ESB is to isolate the coupling between the service that is being called, and the transport medium that is being used. For example, service A may produce a message of type X. Service B may consume a message of type Y. The ESB serves to isolate this interaction, and manage the routing and transformation of messages of type X into messages of type Y, suitable for consumption by service B.

Figure 5 illustrates a notional example of how an ESB ties together multiple applications. All of the applications and data sources at the top of the diagram communicate through the "Routing and Transformation" messaging bus. Service consumers at the bottom of the diagram may use a single standardized service interface that has the effect of invoking multiple back-end services.

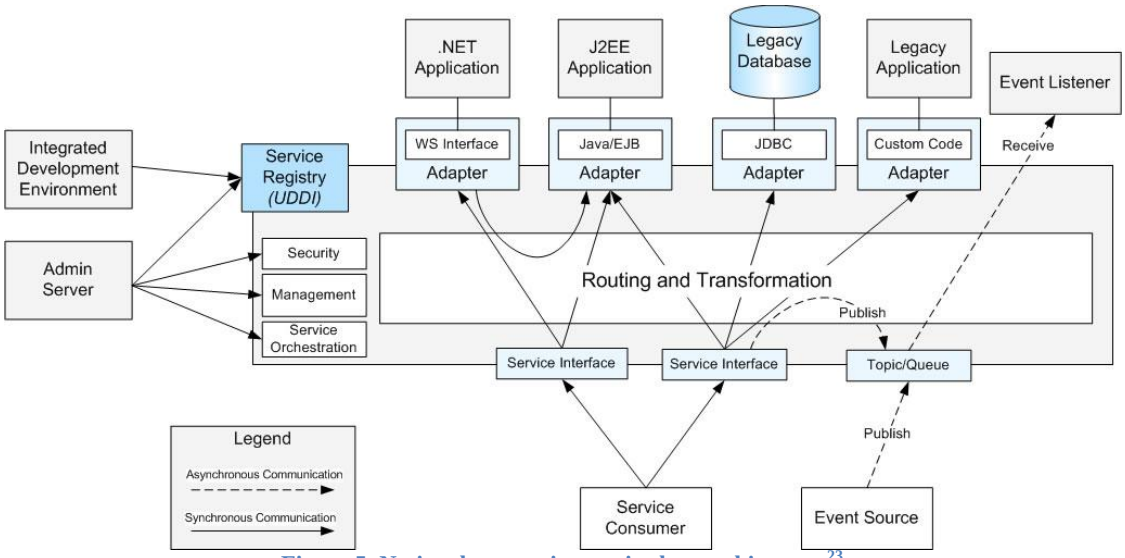

Figure 5. Notional enterprise service bus architecture[23]

The key advantage of ESBs is that they allow for relatively fast and inexpensive integration of multiple systems into a single cohesive whole. They typically provide many predefined service types that are ready to be plugged in, and they are scalable to widely distributed applications, all using a basis of common standards. The disadvantages are that they require a messaging model, for wrapping and handling the content that is passed across the bus, in a way that can abstract away the implementation details of the participating system. This messaging model introduces additional overhead that can be magnified in intense messaging applications.

Another disadvantage of ESBs is that they are often used as a solution to "glue together" multiple systems that are not really integrated. Many systems expose their own idiosyncratic interfaces, requiring point-to-point message translations. ESBs do not fix those issues, but focus on simplifying the process of mediating between otherwise incompatible applications.

Provenance Capture Opportunity

ESBs are a good target for provenance capture because so many different messages and formats all flow through a logically centralized messaging bus. By inserting software that logs these messages as they are sent, rich data can be captured about the generating program or service, the message type itself, and its destination. IM-PLUS has successfully demonstrated that provenance capture using ESBs is a feasible approach to collecting large volumes of provenance quickly and easily. Without requiring modification to any of the underlying software applications or services, a complete map of a complex application's execution can be built for later analysis.

Proof-of-Concept Example: The Mule ESB

The IM-PLUS project has demonstrated the feasibility of provenance capture through ESBs by using the Mule ESB[24] as a test bed. Figure 6 illustrates the architecture of a "Loan Broker" application implemented in Mule that was used as a proof of concept. In this application, a

[23] http://enterprisearchitecture.nih.gov/NR/rdonlyres/3FE0F87B-EB46-4605-8EFA-40B9A9796F1E/0/EnterpriseServiceBusPattern.jpg
[24] http://www.mulesource.org/display/COMMUNITY/Home

customer requests a loan quote from a "Loan Broker" service. That service in turn takes several actions; it requests a credit report from a credit agency gateway application, and passes both the loan request and credit information to a series of lender gateway applications, which respond with specific quotes. Those quotes are then returned to the customer.

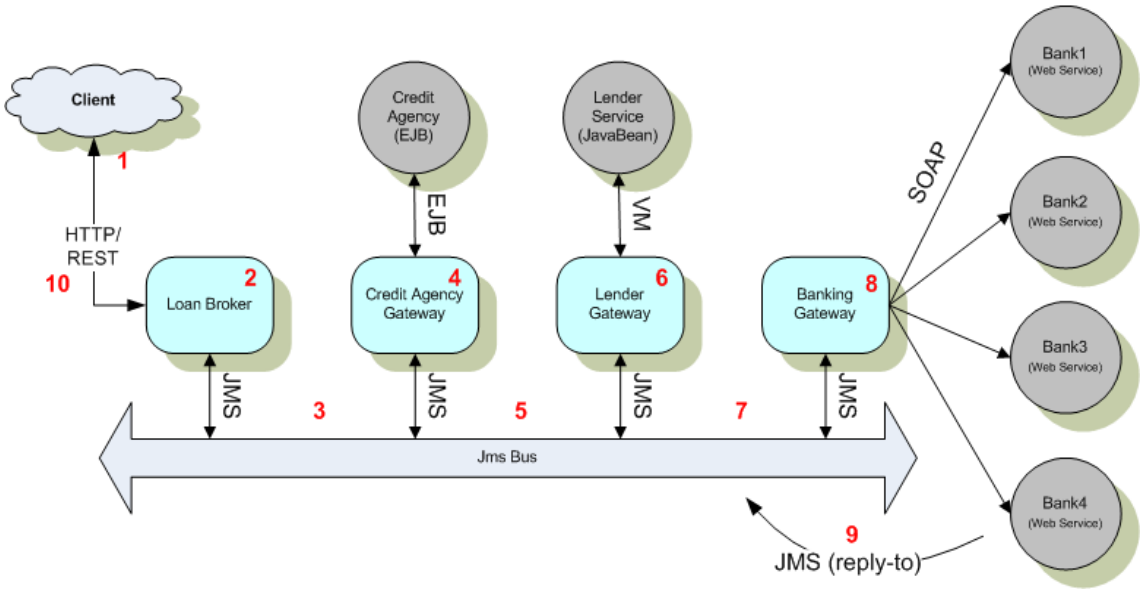

Figure 6. The Mule Loan Broker Application

Provenance capture was implemented by adding "Envelope Interceptors" to each of the services configured on the Mule ESB. Figure 7 illustrates the function of an envelope interceptor. Mule provides a method of triggering a portion of code to execute both immediately before a given service is invoked, and also immediately afterwards. Prior to service invocation, the custom code is passed a reference not only to the input message, but also to the service destination endpoint. After service invocation has completed, a different piece of code is called with a reference to the completed service, and the message that it generated. Figure 8 represents the provenance graph that is logged as the result of a single call to an envelope interceptor.

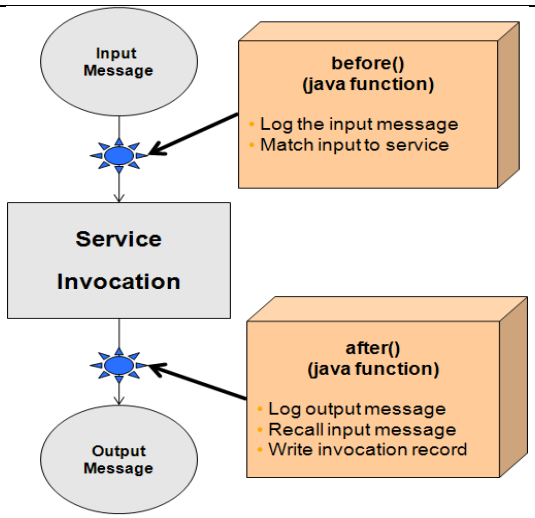

Figure 7. Illustration of the function of an envelope interceptor (execution flow)

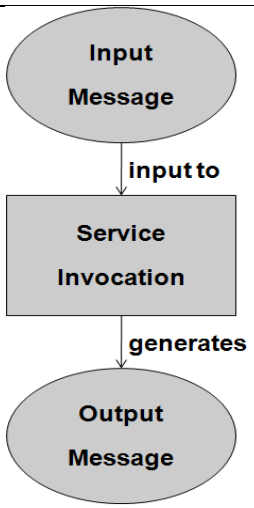

Figure 8. Resulting one-step provenance graph

ESB applications can be viewed as a long series of individual service invocations, all resulting in miniature provenance graphs that look like Figure 8. One of the primary challenges of the provenance logging software is to "connect the dots" – i.e. recognize multiple instances of the same input message, and link the graph accordingly so that many nodes may branch off of the same message that was reused several times. When output messages are sent on to other applications, they are "fingerprinted" with a specialized provenance ID. When that same message later comes in again, (this time in the role of "input message" to the next application) the provenance logging software can recognize that it has seen this message before because of the presence of the "fingerprint", and connectivity is established.

Figure 9 provides a screen shot of a completed provenance capture of the loan broker application. This graph traces the execution path of a single loan quote request that results in two quote responses from separate banks. In the diagram, boxes are used to depict invocations, and ovals are used for data messages passed across the Mule ESB. The data messages passed are in several different formats depending on the service being invoked. This particular graph contains both Java Messaging Service (JMS) queue messages and XML messages. Edges in the graph are labeled with the relationship that two nodes have to one another, according to what was reported by the ESB. Because each invocation is implemented a different way, and there are several different message formats being used, this provenance graph is a good example of how interactions between heterogeneous implementations can be observed.

Figure 10 demonstrates a few sample specific items of metadata that the Mule ESB allowed the provenance plug-in to capture. This metadata corresponds to the "Node Metadata" information described in section 3.3. The protocol, system name, and service endpoint are all included. In fact, the provenance plug-in has available to it any item of metadata about the service or its invocation that Mule itself is aware of.

This raises the question, "how much should be captured, and when should it be captured?" This question will be examined in other sections of this paper, but it can be seen in the context of this

concrete example. Should protocol information be captured as part of the service invocation? Is the information captured below sufficient, or is more necessary? The answers depend on the type of analysis the provenance needs to support.

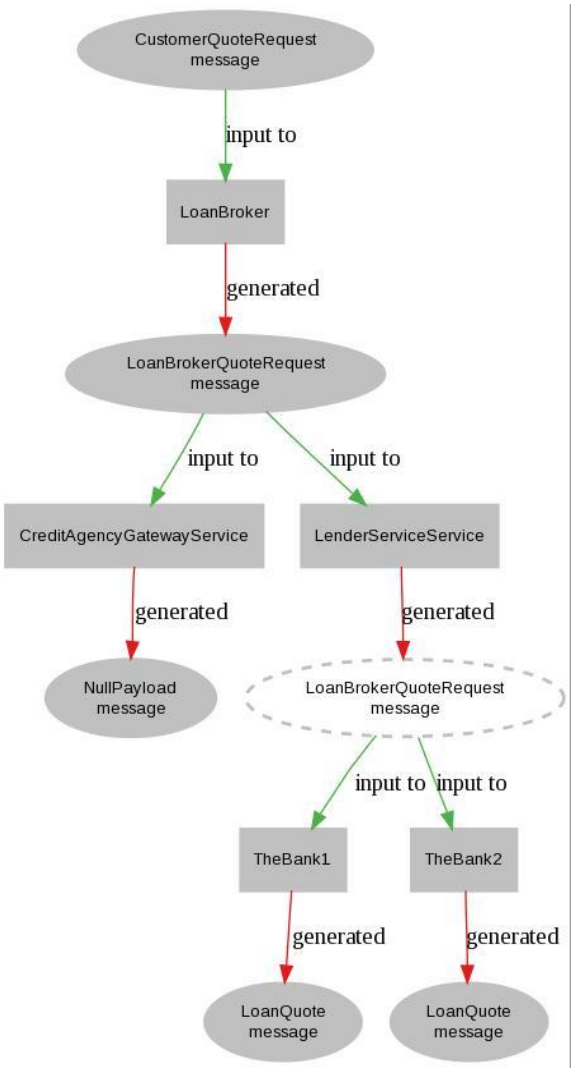

Figure 9. Complete provenance capture for a loan broker invocation

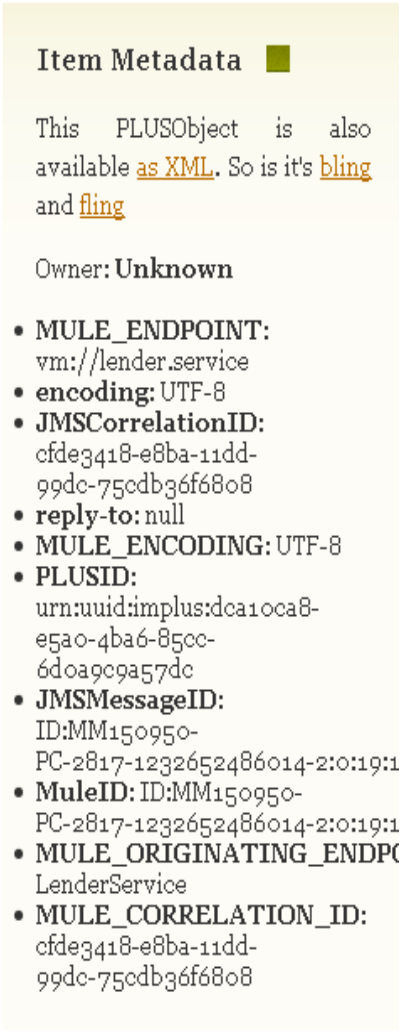

Figure 10. Sample screenshot showing the metadata captured about one node (LenderService)

4.1.2 Business Process Engines

Business Process Engines are software packages meant to execute workflows that are specified in terms of a business process definition language. The Web Service Business Process Execution Language (WS-BPEL) is an example of such a language created for specifying interactions with web services. While WS-BPEL is one of the best known standards, numerous others exist, including the jBPM Process Definition Language (jPDL), and Business Process Modeling Language (BPML). While some ESBs focus on how to integrate existing applications and transform their various inputs and outputs to work together, business process engines typically assume a starting point of standards-compliant services that implement important pieces of business functionality, and then focus on how such services are orchestrated into more complex applications.

Currently, there are numerous commercially available and open source business process engine implementations, including ActiveVOS, Apache ODE, Appian Enterprise, Oracle BPM Suite, jBPM, SAP Exchange Infrastructure, WebSphere Process Server, and numerous others[25].

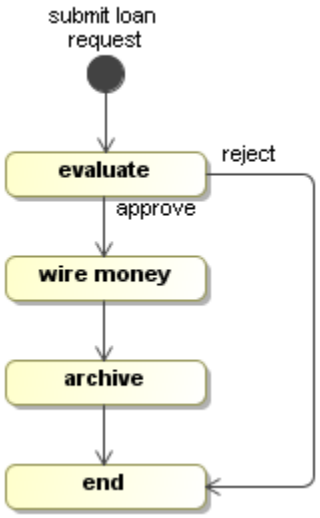

Figure 11. Sample business process workflow[26]

Figure 11 illustrates a relatively simple sample business process that could be specified via BPEL. Such business processes include conditional statements (is customer credit OK?) and potentially even loops which can carry out simple processes repeatedly until exit conditions are met. Organizations can build up complex process descriptions using BPEL. The role of the BPEL engine is to take that process description and execute it operationally, using implemented services and real data.

Provenance Capture Opportunity

BPEL Engines provide the same provenance capture benefits as an ESB: a single point where provenance capture code can be inserted such that all message flow can be captured without modifying any underlying web services. While their method of function is very different from an ESB, the fact remains that there is a single logical execution engine which orchestrates and controls process execution flow.

4.1.3 System Wrappers

This approach involves developing a small "wrapper" application with the same required inputs and outputs of the application that is being targeted for provenance capture. The wrapper application simply accepts requests for functionality, and forwards them on to the underlying application after logging the application request. When the application responds with the outputs, the wrapper returns them with no modification back to the original requester, also after logging the interaction.

[25] Wikipedia: "Comparison of BPEL Engines" - http://en.wikipedia.org/wiki/Comparison_of_BPEL_engines
[26] jBPM "Loan Process" example - http://docs.jboss.com/jbpm/v4.0/userguide/html_single/#processdefinitionprocessinstanceandexecutions

System wrappers are in essence a way of implementing Mule's "Envelope Interceptor" concept for any given application or service. Their primary advantage is that they do not require any modification to the underlying application or service. The main disadvantage, though, is that a wrapper must be developed for almost every service interaction method (or even for each individual service), and that workflows must be re-specified to invoke the wrapper rather than the underlying application. Another issue with system wrappers is that they will typically report one-step provenance (i.e. request, process, response) rather than more extensive and rich graphs. The exercise of linking many one-step provenance graphs becomes the burden of another system. *Because of these reasons, system wrappers are generally an inferior method of capturing provenance to either ESBs or business process engines. However, for environments that use neither, system wrappers are a possible alternative.*

4.2 Application Modification

An alternate strategy is to modify applications to report provenance via a provenance capability's report interface, as described in Section 2. This entails changes to the source code and recompilation of the program in question, and as such is probably the most labor intensive option. The single advantage is that it provides limitless flexibility in provenance reporting, because the application can report things as fine-grained or as coarse-grained as is necessary, and in accordance with the application's internal model of the problem domain. As such, the provenance captured can be precisely tuned to the application's needs.

While application modification is a reasonable capture strategy for applications with both an overriding need for provenance and specialized provenance requirements, it is not sufficiently scalable to be a primary strategy because it is so labor-intensive. For most applications, the ROI will be insufficient to justify the cost. In addition, for many government problems, the application modification strategy is infeasible because access to the application source code is often unavailable due to licensing or contract issues or simply because the software is so old that no one has the code.

4.3 Manual Provenance Capture

As noted in Section 1.6, *manual population of provenance metadata is not a scalable, enterprise capture strategy*; all too often, provenance is simply not collected or, even if it is, it is of insufficient quality to be useful.

Nevertheless, manual capture can sometimes play a useful, albeit limited, role. Typically, this kind of scenario arises when provenance is needed for a database which is curated[27], where the contents of the database change slowly or infrequently. Databases storing the results of scientific experiments, isolated records management databases, and others may fall into this category.

The advantages and disadvantages of manual provenance capture are similar to those of custom application modification; manual capture is likely the slowest and most labor intensive option available, but one that presents endless possibilities for capture of fine-grained, precise provenance information that is tailored to a particular use. Manual provenance capture may be

[27] "Provenance Management in Curated Databases", http://www.eecs.umich.edu/db/files/copypaste.pdf

the only feasible alternative in cases where the processes being executed rely heavily on human interaction (such as group document collaboration). Where there is no automated system that tracks the execution of a process, automatic collection of provenance may not be possible.

Wherever possible, manual capture should be supported by task-appropriate tools. For example, an intelligence analysis toolkit may give users the ability to import documents into a workspace. In such a case, a tool may be able to make reasonable suggestions (e.g., that the document being imported may be pertinent to a product currently being produced). Auto-completion and other automated support are also possible even when capture is primary manual.

4.4 Operating System Observation

Most modern operating systems can be extended with additional software modules that allow application code to run with the same privileges of the operating system itself. Operating systems typically also provide the ability to run memory-resident programs which can observe the execution of other programs on the system. Examples of such programs include intrusion detection systems (Tripwire, Snort), operating system kernel modules, and development tools such as profilers and debuggers. A similar approach can be used for non-intrusive provenance capture. Harvard's Provenance Aware Storage System (PASS)[28] provides an example of this approach, by implementing a program that runs within the Linux kernel and observes provenance as users and programs interact with the operating system.

The advantages of such systems are that they provide a silent and transparent way of gathering as much provenance information as is desired. A downside is that such programs tend to report provenance at an extremely fine-grained level, because the software resides at a very low level in the technology stack. The result may be too much detail and a semantic mismatch between low-level operating system calls and a user-level view of executing processes. (However, it is possible that this low level of capture could have benefits for auditing applications.) In addition, operating system observation only collects provenance on a single system, so when workflows span multiple machines, the challenge still remains of linking the separately created provenance stores.

While the approach has been implemented in the PASS system mentioned above, it has never been applied to real applications. Given our concerns about the low semantic level of captured provenance, we advocate experimentation with operating system observation to assess its utility on realistic problems before attempting to field it.

4.5 Log File Parsing

Provenance information is similar in many ways to the kind of information found in an average program's execution log. Most standard programs have the ability to save some record of what the program did, and what results the program produced. This sort of provenance is extremely constrained, in that it tends to be "one-step" provenance, with little or no linkage back to earlier processes, or subsequent later users of the same data. All the same, because of the ease of access

[28] Muniswamy-Reddy, K.-K., D.A. Holland, U. Braun, and M.I. Seltzer, "Provenance-Aware Storage Systems," *USENIX Annual Technical Conference*, 2006.

to log files, their commonly understood formats, and the sheer volume of data that they represent, they should not be ignored as a potential source for provenance information.

A further reason for considering this approach is the proliferation of scripted programs that are de facto workflow control systems. Frequently, complex workflows are implemented on Windows and UNIX systems in terms of a series of smaller programs held together by a control script that executes each program in sequence, logging the results of each execution to an intermediate log. In these cases, examining the log files of a single (even simple) script may yield rich, multi-step provenance usable for other purposes.

Recovering provenance information from log files involves a log file parsing program which would then use a provenance reporting API to "report after the fact" the contents of the log file according to the provenance model being used. Fortunately, for a given log file format, this approach is usually quick and easy. If care is taken to target the most appropriate log file formats, it is possible to gather a substantial amount of provenance information with relatively little effort.

There are also major disadvantages to this approach. First, the level of granularity of reporting is pre-determined by the application doing the logging, and may often be insufficiently detailed or simply not suitable for a particular kind of analysis. Second, capturing provenance through log file parsing will miss a lot of valuable information that falls outside of the functionality of the logging program. While it may be appropriate for targeted high-value applications, it can only be one tool in the tool box, not the primary approach to provenance capture.

4.6 Summary of Provenance Capture Recommendations

We have described a range of options for provenance capture; they encompass methods that capture either fine-grained or coarse-grained information. The options provided range from low effort to high effort, and vary in the level of control afforded to the organization capturing the provenance. Ultimately, the choice of approaches should be driven by the technical requirements of the project at hand. Still, there are some general observations on how to make good choices that we find apply fairly widely.

For most applications, the overriding concern is to minimize the burden of collecting provenance. If this is not done, one could design an extremely rich provenance model but end up with a very sparsely populated provenance store. Without populated provenance, the rich model will fail to meet most analysis needs.

Minimizing the collection burden means avoiding manual capture and application modification whenever possible. The most promising and general method for low-cost, non-intrusive capture appears to be observation at multi-system coordination points, such as ESBs and business process engines.

However, this general strategy will not meet all application needs, for example, those that require very fine-grained provenance. For such applications, the other approaches described are worth considering. Finally, for primarily manual processes, there may be no alternative but manual

provenance capture. Whenever possible, the burden of doing this should be reduced by providing task-appropriate software assistants.

5 Capturing Provenance: What to Capture

The previous section focused on the "how" of provenance capture. While there are many different approaches to "how" provenance is captured, it is also important to address "what" to capture, and the impacts those selections have on the utility of the system.

The most important observation to start with is that what is captured depends on what the analysis needs are. It is difficult to provide recommendations on what should be captured in the absence of specific requirements. One could argue, "Capture as much provenance information as possible," but this ignores issues of cost—whether in the form of development effort, system performance, or operator effort. One needs an understanding of requirements and use cases that illustrate the likely uses of provenance in order to properly evaluate the trade-offs.

As this paper is focused on a general audience, it avoids recommendations in favor of enumerating the considerations that should be kept in mind while evaluating any specific set of analysis requirements.

5.1 Granularity

> **Summary**: Tracking provenance for the finest granule of data, such as every row in a database, while maximally informative, is also more expensive than tracking provenance at a coarser level, such as at the level of the entire database. The granularity at which provenance is kept impacts usage and system performance.

One of the most important aspects of what provenance to capture is the level of granularity—i.e., how detailed it should be. Figure 12 provides an example of a single simple provenance graph represented in three different ways. On the left side of the illustration, a very simple, coarse-grained account of the "LoanBroker" invocation is given. On the far right, there is a much more fine-grained account of what is happening in this invocation, effectively splitting the small workflow into two parallel streams of execution. All three of these sample provenance graphs represent the same set of actions being executed; the only difference is the level of granularity at which the provenance was reported.

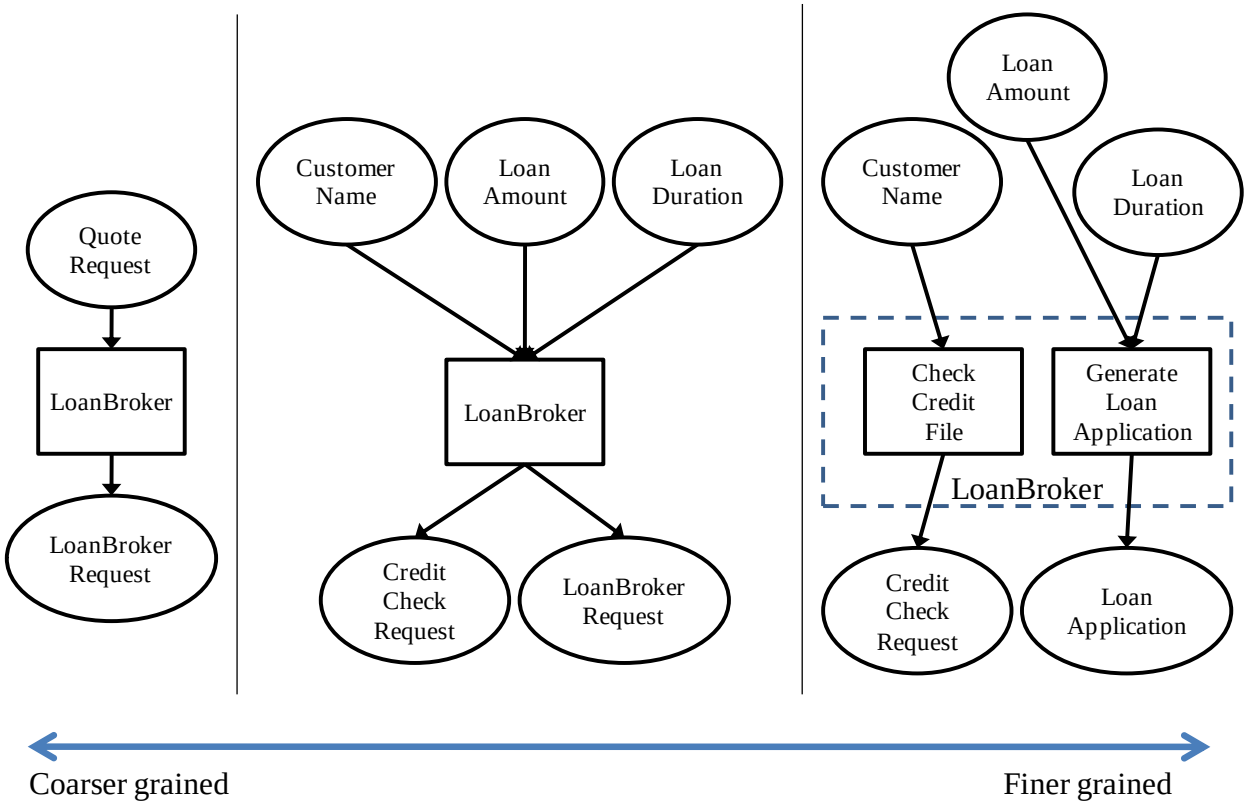

Figure 12. Three different versions of the same provenance graph at varying levels of granularity, taken from the sample "LoanBroker" application discussed in the Mule ESB provenance capture section

5.1.1 Tradeoffs Associated with Granularity Decisions

Figure 12 hints at one of the fundamental tradeoffs in provenance capture; the more provenance information that is captured, the deeper and more nuanced the understanding of a situation can be. The consequence of supporting that deeper analysis is the need for more sophisticated infrastructure to capture the provenance, accompanied by larger storage and processing requirements. On the other hand, simple coarse-grained provenance capture requires very little storage and overhead to capture and process, but it may make certain types of detailed analysis impossible.

For example, in Figure 12, the lefthand provenance graph does not provide any hint about the number of different sources of data required for a loan quote; the metadata logged will likely not make it clear who the customer was, or whether or not that customer required a credit check. Further, if there is something wrong with the resulting "LoanBroker Request" object, it may not be possible to trace it to a problem in a particular input, since they are all lumped into a single object. The righthand provenance graph would likely provide additional detail to answer those questions. Conversely, for simple uses the righthand provenance graph is excessive; and may even be confusing to users whose analysis approaches the problem in terms of a "LoanBroker" process. In short, more provenance information is not always a good thing.

There is an additional granularity issue relating to whether data nodes in a provenance graph refer to specific portions of a data collection (e.g., a particular track in a track database) vs.

treating the collection (e.g., the track database) as a monolith. Even though the abstract provenance capability described in Section 2 supports both options, the collection strategies are often quite different. For example, consider the provenance graph in Figure 13, which accurately tells you that the track database is produced by applying algorithm-27 to a collection of radar reports. However, it tells you nothing about which reports contributed to which particular tracks (e.g., that track-15 was produced by radar reports 3, 11, and 94). While this fine-grained provenance might be useful, there is no way to collect it by unobtrusive observation of systems (for example, by watching an ESB or business process engine). The only way to get this detailed provenance would be from modifying algorithm 27 to report these detailed assertions. As noted in the preceding section, application modification is often expensive, and should not be undertaken without explicit requirements which compel that choice.

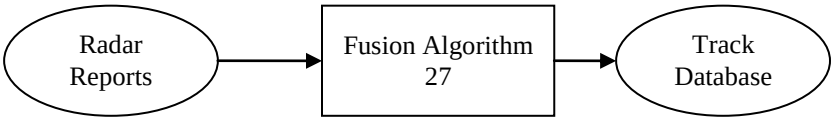

Figure 13. Example of Coarse Grained Granularity for Data Collections

In order to make an appropriate decision about the best level of granularity to capture, an organization must first start with an idea of what sort of analysis is needed. If the goal is simply to provide end-users with a basic ability to trace backwards and identify major data sources, coarse-grained provenance may suffice. If the goal is to provide advanced analysis capabilities to pinpoint particular suspect data values, or to identify where the internals of some algorithm may be breaking down, more detailed provenance capture will make it more likely that the analysis task will succeed.

In many environments, experimentation may be called for in order to determine the appropriate level of granularity to capture. When in doubt, the system administrator may want to start off by capturing as fine-grained information as is easily available for reporting. When in doubt, it is usually preferable to capture too much and later summarize the information, rather than capturing too little and finding dead ends in the analysis process. Once an initial capture is accomplished, the volume and quality of this information can be assessed for practical impact (storage, processing, overhead) and analysis support.

5.1.2 Practical Limitations to Granularity

Provenance capabilities are designed to operate in distributed environments that include multiple partners, who may not agree. Sometimes, a certain partner may be unwilling to expose full details of the processes that they execute. This may be for legitimate security reasons (unwillingness to expose sources and methods), or also for practical system considerations (inability to bear the additional overhead of reporting fine-grained provenance on platforms limited by hardware and software concerns). This often leads to the challenge of "black box reporting," which sometimes can be mitigated by "grey box reporting."

Black Box Reporting

Figure 14 provides an example of "black box" reporting of provenance. On the left hand portion of the illustration, there are a number of different data items and processes being executed, all of which produce a final output called B. This left hand portion represents the reality of what is actually occurring in the system. On the right hand side of the graph, we see what is actually

reported: all of that complexity is collapsed down into a single, minimally informative "Process A".

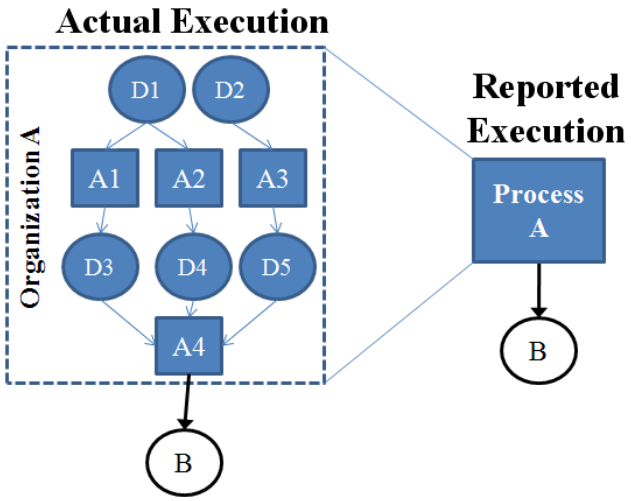

Figure 14. Black box reporting of a complex process

This sort of black boxing may result from organizational boundaries, inability to capture detailed provenance within Organization A, or concern about the consequences of disclosure.

It is important to note that this "black box reporting" is different than a simple view or way of displaying the provenance. "Collapsing" a series of nodes into a single overall process (similar to the figure above) is sometimes desirable, (see A.2 for an example of this "visual clustering") to make the data more granular and easier to view for analysts. The key difference here though is that in "black box reporting", the additional detail was never reported, and doesn't exist in the provenance store. This is different from situations where detail is sometimes hidden from the user to make graphs easier to analyze.

Grey Box Reporting

While not all black boxes can be decomposed, it is sometimes possible to infer more fine-grained provenance information on the basis of coarse-grained reporting. As an example, consider the Mule ESB discussed earlier. In the case of Mule, often messages passed between components are reported as a single, monolithic XML message. For certain key types of messages, the provenance capability can be equipped with a basic understanding of the schema of those messages, in order to introspect into the message and determine that while Mule is reporting it as a single message, in fact the message has 5 distinct components as defined by the schema. In such cases, although the provenance reporting platform (Mule) reports only a single data item, additional hints given to the provenance capability can effectively "unpack" this single data value into a range of different values. Such reported data items might be referred to as "grey boxes", as they provide some basic hints at internal content (such as a schema reference) without explicitly providing full details.

While this is possible, it inherently requires more effort because the provenance capability must be aware of various data formats, and must have additional background knowledge about the domain or problem area. Wherever possible, it is preferable to derive finer-grained reporting

from the reporting environment (such as Mule) rather than building in additional functionality into the provenance capability itself.

5.2 Transparent Translations

> **Summary**: Translations of data from one format to another often introduce important changes to the meaning and use of the data. Provenance capture should attempt to identify and capture relevant translations whenever possible; however, not all transformations can be captured.

Transparent translation of data is one of the larger challenges that arises when capturing provenance in distributed environments. Figure 15 illustrates an example of how an ESB or business process engine may perform many different translations in a pipeline process operation. The fact that these translations are transparent to the user is actually a primary function and selling point of some software infrastructures. This figure illustrates how a java object can take an HTTP message as an input, and how the JMS message that it produces can be consumed by a web service (that would normally take XML as input).

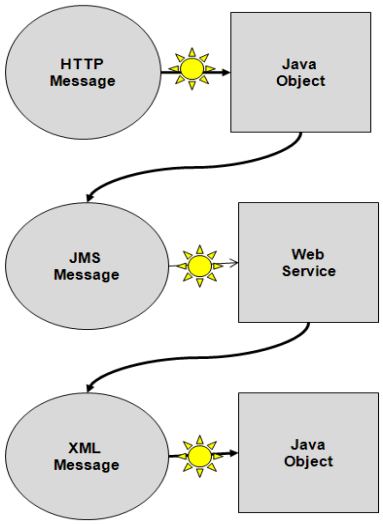

Figure 15. Translations between multiple data formats, signified with yellow starburst icons

Each of these translations has an unspecified effect that may or may not affect the quality and reliability of the data flowing through the system. In the case of lossy data conversions (i.e. a conversion where the target data format is less expressive than the source format), data is typically lost or altered out of necessity. In software infrastructures where such translations are not modeled as processes themselves and hence subject to provenance capture, this data loss may be invisible in the resulting provenance graph.

Figure 16 illustrates this point further, introducing the concept of message routing. In the upper portion of the diagram, the data labeled "A" is sent to both process p1 and process p2. Because of the intervening translations, though, process p1 and p2 see different versions of the same data. The resulting provenance capture, however, does not make this distinction – node A is shown as a direct input to both p1 and p2. While this perspective on the execution is not incorrect, it is also not the complete story of the execution.

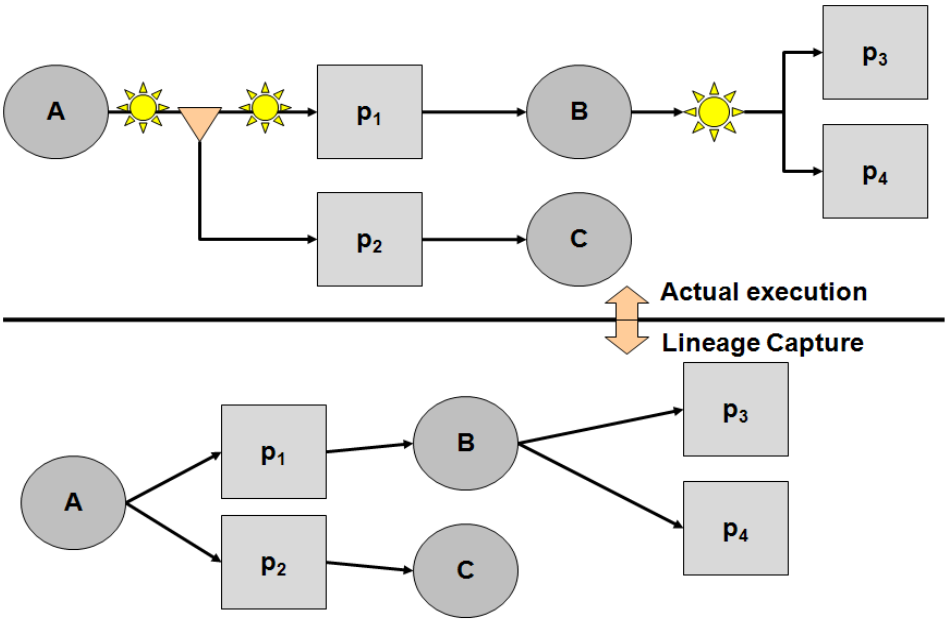

Figure 16. Illustration of the difference between actual execution and provenance capture

Many of the issues with transparent translations can be easily addressed, if they are properly identified. For example, the Mule ESB provides a mechanism to translate messages as they are shuttled between services. Translating messages in this way makes the transformation invisible to provenance capture. On the other hand, Mule provides a way of specifying that translation as a process unto itself, with transitions to other processes. When modeled in this way, provenance capture is aware of the translation as just another standard service, and captures it appropriately.

Where possible, system designers should take care to identify relevant translations that substantively affect data content, and orchestrate their applications to use those translations as services, rather than as transparent software modules invoked during service transitions.

5.3 Choice of What to Capture

> **Summary**: A provenance capability should not try to capture everything. Capturing all possible provenance information is generally not feasible nor suitable to the particular requirements of most applications. There are many reasons why an organization would choose to capture much less provenance than what it is capable of capturing. Usage requirements should be used to filter provenance capture opportunities down to only what is necessary for the task at hand.

The choice of how much provenance to capture requires consideration of many constraints. This section enumerates a number of constraints and considerations to help guide the scoping decision of how much provenance should be captured in any given scenario.

5.3.1 Constraints on Provenance Capture

Capturing provenance has a small but real performance cost associated with it. In service-oriented environments, possibly the largest cost is potential network latency associated with capturing provenance. Provenance capture may also be inappropriately applied on time-critical

processes, or those that need to execute many hundreds of times per second, slowing those processes by introducing additional capture overhead.

In situations where the workflow is small, short-lived, and repeated in very large numbers, capturing all available provenance information can result in provenance graphs that are substantially larger than the base data they represent. Organizations need to carefully consider the use of the captured provenance before deciding to capture this kind of information. Additionally, systems with strict memory, disk, or processing restrictions may choose to avoid provenance capture, or to have a different system with fewer constraints perform the capture work. Such performance restrictions may influence the choice of architecture and deployment options.

Even within high-level workflows, there are often repetitive or "looping" components which serve to increase the size of a provenance graph substantially, without necessarily adding additional value for the analyst. Figure 17 illustrates an example; a module named "d" executes 1,000 times in sequence, yielding a provenance graph that is a single long chain.

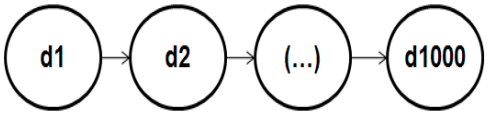

Figure 17. A long-chain "loop" represented in provenance, suitable for abstraction into a single node

In such situations, it is often preferable to avoid collecting information on individual invocations of "d", and instead abstract it to a single "d" node, hiding details of exactly how "d" executes. (The University of Pennsylvania's Zoom project[29] has demonstrated such a capability.) In most situations, this will reduce the size of the graph with no loss of information, since almost everyone in the user community knows about this linkage and assumes it.

Aside from "loops", other situations arise that might suggest capturing less provenance information rather than more:

- *Implied inputs*, (e.g. bank interest rates always use the Federal Reserve rate as an input, so it may not be necessary to capture that particular detail)
- *Processes whose internal flows are well-known*, and where additional detail is unnecessary
- *Data whose lifetime is short* (i.e. transactional details) to the extent that the base data is not expected to exist during later inspections of provenance
- *Locality of focus*; local users may only be interested in the local aspects of an application. There may be many steps necessary to prepare inputs to that local application, which can be abstracted away or removed because they are too remote.
- *The implied trust principle*. The purpose of provenance is to support the consumer's trust decision in data. If there is a process or data that is trustworthy or authoritative by fiat (e.g. the Federal Reserve interest rate, or official statistics), such provenance may not be necessary.

[29] Cohen-Boulakia, S., O. Biton, S. Cohen, and S. Davidson, "Addressing the provenance challenge using ZOOM," Concurrency and Computation: Practice and Experience, vol. 20, 2008.

- *The system trust principle*. Producers will not report provenance unless they trust the provenance capability to respect their authorization and disclosure rules. It may be desirable to omit some sensitive provenance information.

5.3.2 Unanticipated Uses of Provenance

Most or all of the constraints mentioned in this section limit provenance capture in part because they assume that provenance capture should be tailored to local requirements. For example, in the "implicit trust principle" there is an assumption that everyone trusts the same authoritative data sources. In abstracting well-known processes and avoiding capturing the details, it is assumed that all users understand that process.

However, provenance may have unanticipated uses which could be severely hampered by these assumptions. Provenance captured to meet local requirements may not be suitable for a non-local, unanticipated use. In practice, a balance must be established between two extremes:

- *Broad capture*: Capture as much information as possible for a potential (hypothetical) unanticipated user, even if this broad capture is onerous for the community of anticipated users.
- *Narrow capture*: Capture only what's necessary for the local community's pre-identified needs. Ignore the possibility of broader requirements.

A provenance capture strategy should steer between these two extremes by taking into account the likelihood of unanticipated uses and the marginal costs of additional provenance capture.

6 Delivering Provenance to Users: Architecture

Once provenance is captured, there are two basic ways to deliver it to users: as its own service or as an augmentation of an existing service. Additionally, organizations have a third option of federating existing stores that are available. In essence, this third approach is to deploy provenance capabilities in whichever places are convenient to meet local needs, and to tie them all together through a federation scheme.

6.1.1 Provenance as a Service

Deploying a provenance capability as a service involves making the provenance store accessible on a broad network to all authorized users, and providing an abstract service interface (such as through the use of WSDL-described web services) to users allowing them to query the provenance store. Figure 18 provides an example of how a legacy application can report provenance or a user can issue queries against the provenance service and receive responses.

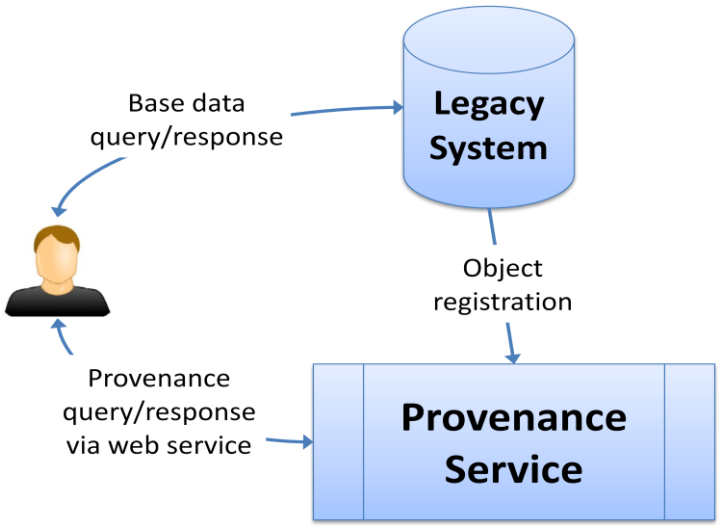

Figure 18. Deploying a provenance capability as a simple service

If the user needs base data, they do this by directly accessing the relevant legacy system. The legacy system in turn uses the provenance service to register objects and record provenance as it happens.

There are several advantages to this style of deployment. The first is modularity; deploying a provenance capability in this way makes it a module that any application can later reuse. This is highly desirable from an enterprise perspective. Deploying provenance once as a service promotes reuse. A second benefit is simplicity; by restricting the user's interaction with the provenance service to provenance-only queries, it is known ahead of time that the response will be able to satisfy the query.

There are, however, several disadvantages. Sometimes, users do not need provenance-only questions answered. They may instead need to gain access to provenance data as part of a defined analysis thread that may also require underlying data and information from other sources as well. Without wider orchestration of the provenance service with other components, users may not be able to perform complex analysis without many steps. For example, if a user needed to find the source information for all equipment data that contained information on a particular type of aircraft, such a task would be broken into multiple steps; discovery of relevant assets, searching their content to verify that they contained information on the relevant aircraft, and connecting the results of that search with the provenance query results. In this way, local analysis requirements may conflict with the "enterprise reuse" perspective. Many local users of provenance may want to log and exploit provenance only for very narrow purposes. In these cases, they may choose to use provenance as a way of augmenting something they already have in place. The enterprise reuse perspective would typically suggest that the provenance capability should be offered as a generic component to everyone within the enterprise; deploying it in this way may make it more difficult for local needs and narrow purposes to be met.

6.1.2 Provenance as an Augmentation of an Existing Service

The second potential deployment scenario is to use the provenance capability as a way of augmenting an existing service. Figure 19 illustrates how such systems might communicate. In

this example, the user does not interact with the provenance capability at all, but rather interacts with a value-added analysis service (labeled "mixed query service" below). The intent with this architectural configuration is to provide users with a single point of interaction that understands the business needs behind the type of work they are doing. That single point of interaction (the "mixed query service") then draws resources from whichever systems are necessary to complete the task.

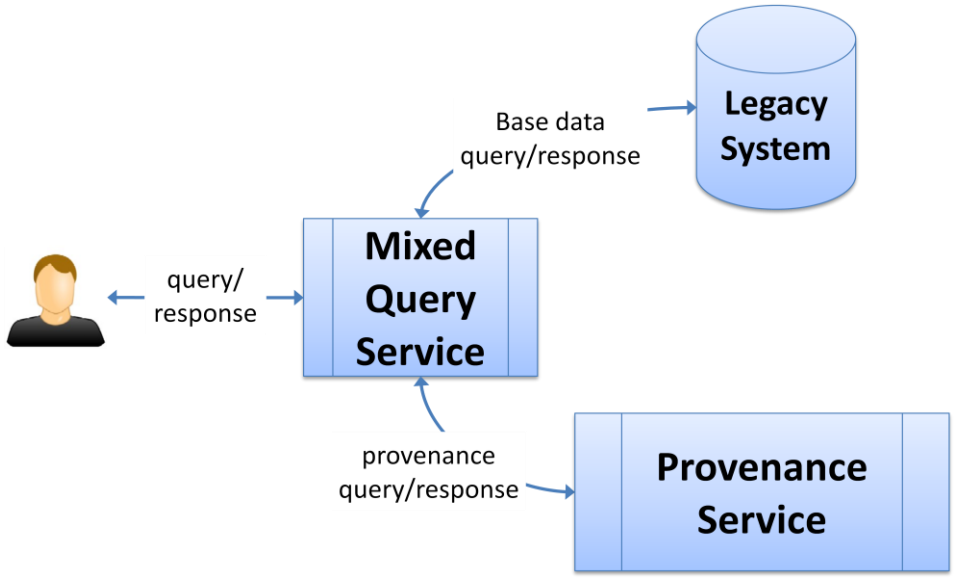

Figure 19. Provenance being used as part of an augmented analysis service

In a deployment scenario where the provenance capability interacts directly with a mixed query service, the interface need not be specified in terms of standards-compliant web services; in fact the software describing the provenance capability may very well be directly incorporated into the mixed query service. To illustrate this idea, Figure 20 shows how a provenance store or service could be tightly coupled with a much larger application.

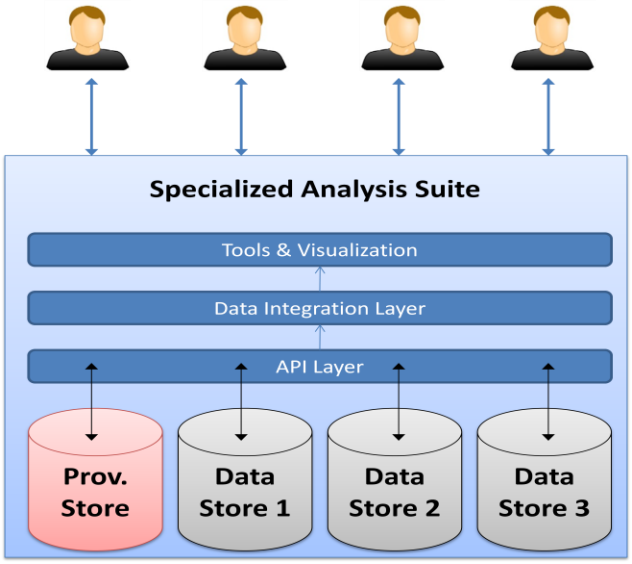

Figure 20. A provenance store tightly coupled with an analysis application

The primary advantage of this approach is to abstract the capability away from the user, and present the user instead with an interface that is customized to their analysis need. From the user's perspective, provenance is just one source of information that is taken into account as part of the analysis. Additionally, because the provenance capability is hidden from the broader information sharing environment, there are many more interface options that can be considered. Rather than having an open interface which all information sharing partners use, the interface may be designed and tuned for the convenience of the mixed query service.

The primary disadvantage of this approach is that it "buries" the provenance capability inside of another system, so that only users of that system have access to the provenance capability. This approach is in conflict with larger enterprise goals of reuse, but may be appropriate locally for systems that have well-defined requirements.

The third and final approach is an attempt at reconciling the two previous approaches via federation.

6.1.3 Federation and the Long-Term Perspective

Inevitably, when new stores of data and metadata are implemented, organizations must eventually address what to do when such stores begin to proliferate. Provenance capabilities are no different, and the ability to link provenance across multiple stores will likely be important to the overall utility of the data. While it might be useful for the Army to have a provenance capability and for the Navy to have one as well, separating the provenance stores in this way creates a situation where a Navy analyst may not be able to analyze the flow of data from the Navy to the Army, because at least some of that provenance is stored on an Army system.

The general trend for addressing such issues has been the introduction of federation techniques. For example, commercial products abound for federating database management systems. The DoD Metadata Registry (MDR) provides a government example. The MDR has implemented the ebXML federation specification. Any metadata registry that supports that specification may

federate with the DoD MDR, so that users who access any system in the federation have access to all metadata in all systems participating in the federation.

At present, there are not enough operational provenance stores for this issue to have arisen. While such federation work has not yet been done, should this situation arise in the future, existing federation strategies (such as the ebXML approach) should prove appropriate for federating provenance stores as well.

7 Delivering Provenance to Users: Supporting Query

Provenance query is the process of answering user questions on the basis of the information available in the provenance graph—that is, the constructs described in Section 3. This section describes the different types of queries a provenance capability can support and closes with a brief discussion of mixed queries over both provenance and base data.

7.1 Supportable Queries

7.1.1 Navigational Queries

The simplest queries are navigational in nature; the user starts with a particular node of interest, and wants to see the immediate upstream and downstream linkages in the graph. In many kinds of unstructured analysis, the user may not know what sorts of detailed questions need to be asked until a baseline context is established for the node, determining what the node is, and how it relates to nearby nodes.

One of the main motivating use cases for provenance is in helping to support a consumer's trust decision in data. For example, when a user is presented with a new data asset, they may choose to inspect its backwards provenance to see what the basis for that data is, in order to determine whether the data is trustworthy for their purpose. This kind of unstructured "investigation" of backwards provenance does not require sophisticated query, but simply the ability to reconstitute provenance graphs from a data store.

Impact analysis also requires navigational queries. For example, if one suspects a piece of data has been compromised (e.g., because of a data modification cyber attack), one would inspect the forward provenance to see what downstream data may have been based on the compromised data.

7.1.2 Path Queries

Once some basic context is established, the user may wish to ask path-oriented queries about the graph. Rather than being interested in any direct path between nodes, questions become more general, such as:

- Is the "Analysis Report" node in my backwards provenance anywhere?
- Is this particular node upstream of any invocation owned by my organization?

In this way, the semantics of individual edges are often collapsed into the simple notion of "something connected to something else" via some chain of interaction. The user is later left to investigate further what the specific path of interaction was, and what its impacts might be.

However, there are limitations to the sorts of path queries that can currently be asked. For example, matching paths in a graph of a particular structure, but arbitrary length, is a current area of research.[30,31]

7.1.3 Update and Delete Queries

At present, there are no known provenance systems that support "update" and "delete" style queries where information is either modified or removed from the provenance system. This is primarily because the nature of provenance is that the data is unchanging; provenance records capture what happened, and that record does not change with time. However, as we note in Section 2, the ability to annotate a provenance graph is critical to many uses of provenance. So, while the underlying provenance graph is usually immutable, the ability to add, change, and delete annotations is essential.

Other than annotations, the only needs for update and delete are special cases, such as administrative correction due to a software error, or deleting large numbers of provenance records from the active data set if some period of time has passed or they have been archived in another location. In such situations, the provenance capability does not need to provide the native capability to execute "update" and "delete" queries, but these types of queries can be delegated to the storage layer that hosts the provenance capability. For example, the IM-PLUS prototype uses a MySQL relational database to persist provenance information. While the IM-PLUS prototype itself provides no functionality to update or delete provenance information, the underlying database still does, in the event that administrative correction needs to be done.

7.1.4 General Graph Query

Aside from the previous more simple types of query, there is the general issue of graph query, where users may ask to retrieve information on the basis of arbitrarily complex predicates that test both individual node values (such as "creator") and also types of node interaction (i.e. number of outputs, edge labels, path length, and other characteristics).

There are a number of issues involved with supporting full graph query, which are not specific to provenance graphs. The SPARQL[32] query language (and other similar graph query languages) perform most or all necessary graph query functions, and can be used to query provenance information. Given this, we do not generally recommend that a provenance capability develop its own graph query support. Instead, when full graph query is needed, the provenance store should be exposed such that it can be consumed by existing tools. For example, if an organization is capable of writing and executing SPARQL queries, an RDF "view" of a provenance graph would be most suitable to allow for reuse of existing in-house query tools. We have demonstrated such a capability within the IM-PLUS prototype; while the underlying data store is a relational database; we have provided a view through which graph queries can be accommodated via a SPARQL query endpoint. Tools (such as the D2R server) are available that permit exposing relational databases as RDF and other formats, facilitating this process.

[30] "Choosing a Data Model and Query Language for Provenance", DA Holland, IPAW 2008 outlines a proposed language called the "Path Query Language" (PQL) which supports some of these features.
[31] http://esw.w3.org/topic/SPARQL/Extensions/Paths lists current (provisional) extensions to the SPARQL graph query language to handle such capability.
[32] http://www.w3.org/TR/rdf-sparql-query/

At present though, graph data storage and query is not widespread, with the exception of the semantic web and RDF community. There is real and readily available software that can facilitate asking quite complex questions of provenance graphs today, but an additional interface on top of the query language will be useful for users to construct basic questions without requiring deep knowledge of the graph query language. This is work that has not yet been done in practice for a provenance capability.

7.2 Future Direction: Mixed Data and Provenance Query

Future users of a provenance capability will likely want to ask queries that freely mix provenance and base data terms. There are many relationships between the two that are likely to yield very useful information. For example:

- How many reports marked "low confidence" were based on upstream data where the temperature sensor gave a reading higher than 100F? (I.e. is there a relationship between a specific input and what we know are bad outputs?)
- We have determined that there is no submarine in this given area. What were the specific sonar buoy readings that led us to conclude earlier that there was a submarine in this area? (I.e. how did we make that mistake?)

Providing answers to these questions can be challenging. Because the provenance capability does not store the base data, it must be dynamically queried in some way from the source that does store it. That in turn requires some system or service to be able to query both provenance and base data, and combine the results in a meaningful way.

Exactly how this could be done depends largely on the deployment scenario that is chosen for provenance. Referring back to section 6.1.2, it may be easier to implement this approach if the provenance capability is tightly coupled to an existing analysis framework, since the size of the enterprise is pre-determined, and the analysis framework provides a pre-existing integration approach for the various data sources, with centralized administrative control.

In a service-oriented environment, mixed query would likely be implemented as a new "Provenance+" service. Such a "Provenance+" service would be more complex. It would need to maintain a list of mappings from those who asserted provenance back to the source systems that provide the corresponding base data. The "Provenance+" service would need to query the provenance capability, query the appropriate source system, and then integrate the results in a way that is meaningful to the user asking the query. Such an arrangement is far from trivial; given the data representation and technology heterogeneity that is likely present.

Existing Enterprise Information Integration (EII) tools as well as more recent "Mashup Servers" are likely applicable in this area; provenance should not be treated as a fundamentally new kind of information with different constraints, but can be thought of as simply another data source that needs to be integrated with the rest of available enterprise data sources.

8 Conclusions

This report has described lessons learned on the IM-PLUS research project about practical steps organizations can take toward a provenance capability. We proposed a general data model for provenance systems that can easily be extended to support application-specific requirements. In addition, we discussed alternative approaches for capturing and using provenance information and, where appropriate, made recommendations and examined trade-offs.

So what is the way forward on provenance? As we have observed, many organizations now recognize provenance as an important capability. This recognition will only grow as government organizations make continued progress on information sharing. However, as noted above, the technology to support provenance is still immature, and the concepts of operations for deploying provenance are even more so. Given this, we do not yet advocate rolling out sweeping, enterprise-wide strategies for provenance. Instead, the time is now ripe for pilot programs in application domains that have strong needs for provenance. Such pilots will provide valuable lessons that will enable our sponsors to be much smarter about deploying provenance more broadly.

In addition, we believe that provenance remains an important area for continuing research. One topic of particular importance to MITRE's sponsors is how to harness provenance to enhance users' decisions about how much trust to place in data that comes from far-flung sources. Key challenges include: managing provenance in cross-boundary information sharing environments and enabling efficient query across distributed, autonomous provenance stores. Another need is for mission-oriented research in critical application areas, such as exchange of electronic health records and intelligence, surveillance, and reconnaissance (ISR).

9 Acknowledgments

The authors thank Dr. Vipin Swarup, Dr. Julie DelVecchio Savage, and Dr. Steve Huffman for their support of this research. In addition, we thank Dr. Arnie Rosenthal, Dr. Michael Morse, Chris Wolf, Professor H.V. Jagadish, Professor Margot Seltzer, and Uri Braun for many valuable discussions on provenance.

Appendix A. Sample Lineage Graphs and Discussion

A.1 Display of Partial Lineage Graphs

Frequently, it is not practical to visualize an entire lineage graph all in one image. There may be too many nodes to view comfortably, or their inter-connections may be too complex.

As a result, we have adopted the simple convention of showing partial lineage graphs with (+) symbol added to nodes for which there is more information available. In this case, both the "Historical Disease Data" and "Trend Model Simulator" nodes have more information that is not being shown.

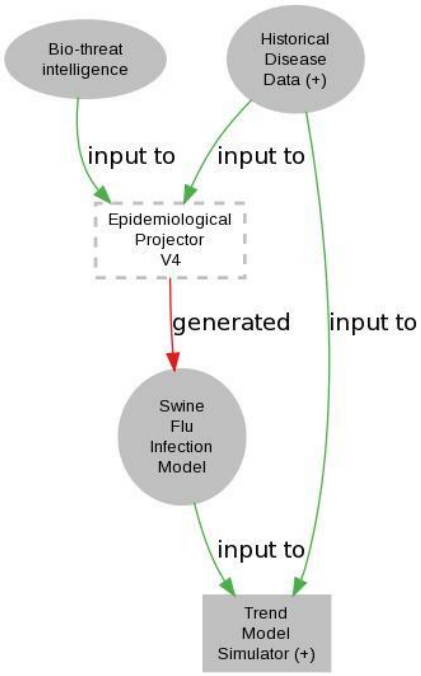

Figure 21. Example of a partial lineage graph with abbreviation markings

A.2 Visual Clustering

Often, it is useful to visualize lineage graphs according to "clusters" of useful information. In these two examples below, all nodes have been combined into clusters. Both of these images use the same base lineage graph, and simply represent different views of the same information.

Figure 22 displays the graph clustered by function, collapsing both invocations and their outputs together into a single cluster. Figure 23 demonstrates clustering by owner of the node, simply drawing boundary boxes while maintaining the overall shape of the graph. Figure 24 clusters the same information by the type of node, radically changing the display.

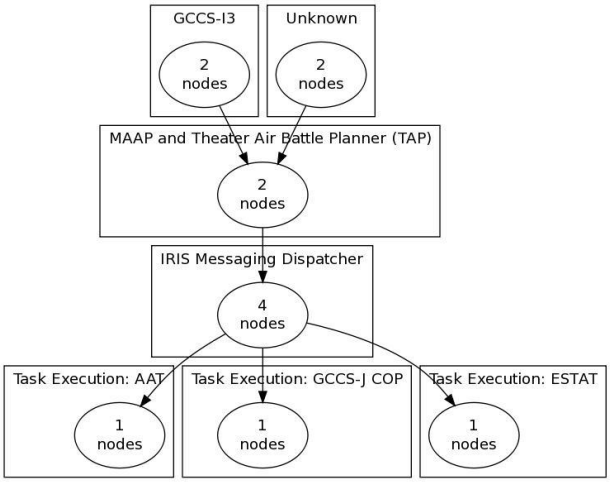

Figure 22. Sample TBMCS workflow clustered by function

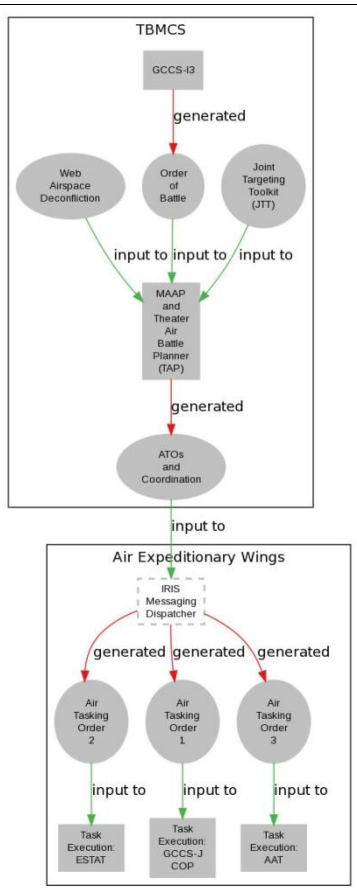

Figure 23. Sample TBMCS workflow clustered by the owner of the lineage node

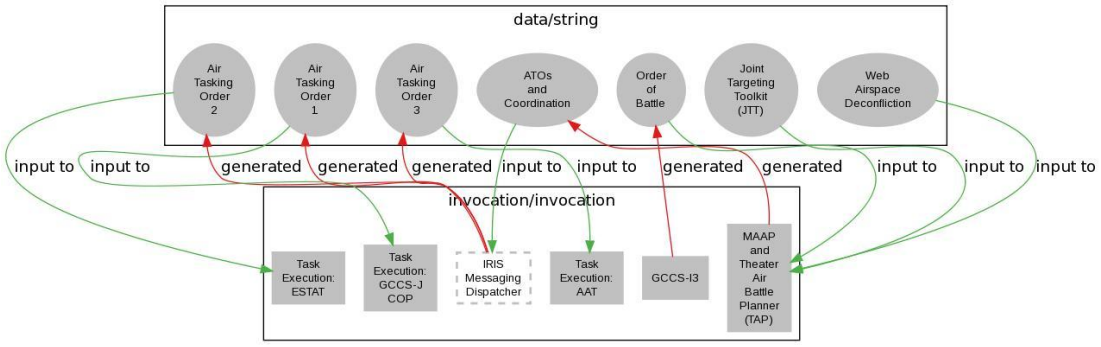

Figure 24. Sample TBMCS workflow clustered by node type

Appendix B. Additional Government Examples

The introduction described the DoD Net-Centric Data Strategy's description of the need for lineage. We now present a few more representative examples.

In 2005, when the DoD released the Implementation Guide for Communities of Interest, the DoD CIO again specifically reiterated the need for lineage and pedigree information.

> *"A consumer that can locate, access, and understand a particular data asset, will want to assess the authority of the data asset to determine whether the contents can be trusted. Promoting trust focuses on identifying sources clearly and associating rich pedigree and security metadata with data assets to support the consumer's trust decision*[33]*"*

The Implementation Guide additionally provided a definition of "asset pedigree metadata" that allows us to see the likeness between the "lineage" discussed in this paper, and the goals of DoD leadership:

> *"**Asset pedigree metadata**. The source and lineage of an asset are its pedigree. The purpose of the pedigree is to enable consumers to determine whether the asset is fit for their intended use and to enable them to track the flow of information, its transformations, and modifications, through assets. Notional metadata describing an asset's pedigree would include creation date, modification date, processing steps (including methods and tools), source and author (if known) status, and validation results against a published set of constraints*[34]*"*

The need for pedigree and lineage is also noted in component and agency data strategy documents, such as the Navy's PMW120 data strategy:

> "*Data Pedigree, both origin and process, is an important aspect for data quality control. Data origin pedigree tracks the flow of data over time. Currently, the data asset inventory only tracks data producers one level back. Data process pedigree refers to the quality of the analysis, manipulation, or fusion of data. Data pedigree attributes shall be part of the overall metadata structures. Details about data pedigree will be addressed by the Data Governance Board.*[35]"

[33] DoD 8320.02G section C4.5.1.1.
[34] Section C4.5.2.2.1 of DoD 8320.02G
[35] PMW120 Enterprise Data Strategy Section 3.1.4. PMW 120's mission is to provide net-ready intelligence, meteorological, oceanographic, and information operations products and services that enhance battlespace and global maritime domain awareness to support warfighting forces and other users of national interest.

