Capturing Provenance in the Wild

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

The MITRE Corporation
{dmallen,achapman,bblaustein,seligman}@mitre.org

Abstract. All current provenance systems are "closed world" systems; provenance is collected within the confines of a well understood, pre-planned system. However, when users compose services from heterogeneous systems and organizations to form a new application, it is impossible to track the provenance in the new system using currently available work. In this work, we describe the ability to compose multiple provenance-unaware services in an "open world" system and still collect provenance information about their execution. Our approach is implemented using the PLUS provenance system and the open source MULE Enterprise Service Bus. Our evaluations show that this approach is scalable and has minimal overhead.

Keywords: provenance, capture, distributed systems.

1 The Challenge of "Open World" Provenance Capture

Provenance, or the history of information, has garnered interest in government, commercial and scientific circles. However, provenance systems will only become ubiquitous when it can be easily captured in the heterogeneous, distributed environments typical of most real-world enterprises.

In contrast to this need, current provenance capture techniques assume a closed world – a contained environment about which the provenance system has considerable knowledge and control. In application-embedded provenance, capture is limited to the data and processes within a particular application [4, 5, 7]. Workflow-based systems, such as [2, 8, 12], can only capture provenance for events that occur within that workflow system. More generic provenance management systems [3, 6, 13] provide a provenance reporting interface; however, because provenance is not a central feature of most applications, the incentives do not exist for them to report provenance. Operating system based systems [11] only capture provenance within a particular machine and, as a result, fail to support distributed interactions across heterogeneous environments. Also, OS-based provenance is usually at too low a semantic level to help users understand the business processes that have acted upon their data. Even the most "unplanned" provenance capture system today, for user-created mashups [7], assumes the applications used in the mash-up are provenance aware and report metadata in a particular format.

All these approaches assume a single system controlled by a central provenance collecting entity. Among our U.S. government customers though, it is common for data to flow across organizational boundaries and for each autonomous stakeholder to use and transform data with their own applications. Therefore, provenance capture must cross system and organizational boundaries. While these systems often expose some interface, their implementation technology is often unknown.

We advocate for a solution that does not require system-invasive strategies and also does not restrict the user's choice of applications. To be useful in an "open world", the solution must capture provenance 1) across multiple systems with no assumption of control over those systems, 2) from legacy systems that are not provenance aware, and 3) at the level of application interaction, not at the level of protocol interaction or foundational technology stack (i.e. OS, filesystem).

2 Provenance Capture at Distributed System Coordination Points

Among MITRE's customers, there are often *points of coordination* in the interactions among distributed, heterogeneous systems. Popular examples include enterprise service buses, business process execution engines, and proxy servers. These coordination points present a previously untapped opportunity to log provenance that spans systems and organizations without requiring application modification.

To explore the feasibility of capture at distributed system coordination points, we created a provenance capture module using an enterprise service bus (ESB) which addresses the requirements described above. An ESB is a tool for integrating multiple applications with different messaging specifications, and for specifying the way in which they interact with one another (the implicit workflow). The popular open source ESB MULE [10] provides built-in support for automatic message routing and translation between different technologies such as Java Messaging Service (JMS), SOAP, etc., greatly simplifying communication among services in a distributed, heterogeneous environment. An ESB is a perfect point for provenance capture in the wild: it is a point of coordination among multiple distributed systems.

By tying capture into a service such as MULE, it is possible to capture provenance of implicit workflows—as opposed to pre-planned explicit workflows—across disparate, autonomous systems. We implemented our approach in the MULE Capture Agent (MCA), which uses the provenance reporting API of PLUS [3], a provenance manager with a model similar to OPM [9]. MCA also uses MULE's "Envelope Interceptor" interface, which allows inspection of the ESB's state both immediately before and after a service is executed. When MCA encounters a new object, it records the object in the provenance store and "tags" the object with a unique identifier. This identifier is carried as the message flows throughout the system, permitting previously unconnected single-step provenance to be linked into meaningful provenance DAGs.

MULE and other ESBs provide a large amount of metadata relevant to service invocations (e.g., a given request came from Firefox or the time of the response was 09:07:23). MCA can access this metadata and report it to PLUS whenever it is anticipated that future provenance queries might require it. In addition to message metadata, some applications that leverage provenance may also require excerpts from the underlying message payload. For example, latitude and longitude may need to be extracted if future provenance queries might include location predicates ("show me the derivation of all today's situation reports within 20km of Port-au-Prince"). In such cases, the ability to peek at the message payload is required. MCA uses several techniques to introspect into messages, including *reflection*, a java ability to examine the runtime behavior of applications in the VM. Whether the information comes from message metadata or is extracted from the message payload, PLUS manages it using its extensible facility for attaching attribute-value pairs to provenance nodes.

3 Evaluation

We tested[1] MCA on two different workflows: LoanBroker is a standard MULE test scenario, and CoTLooper is a scenario that uses Cursor on Target (CoT) [1] messages. CoTLooper is a simple test that provides an example of operational messages containing real and necessary metadata such as latitude, longitude, entity identifiers, and other information used by several tracking and sensor-fusion systems.

Figure 1 shows the average time required to capture one node using the MULE capture mechanism. For this experiment, we used LoanBroker, and ran through 1000 loan requests. Each loan request requires checking with a credit bureau and requesting candidate loan quotes from 3 different banks. We measured the time to invoke the provenance capture function and log the information. While there is a high startup cost because of connection pooling, since MULE is typically used for high-volume and long-running distributed systems, the *more accurate capture times are found to the right of the graph*, where the number of interactions dwarfs the number of connections. In a system where several hundred messages are sent per second, the average per-transaction cost of provenance capture is very low.

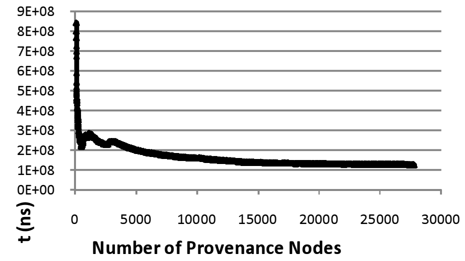

Fig. 1. The average time to log provenance within MULE over time

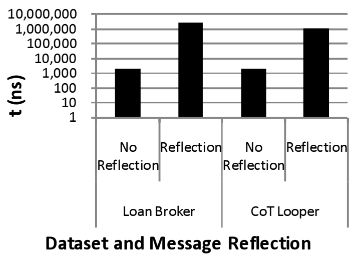

Fig. 2. Message Reflection effect on Message Capture Time

Figure 2 shows the time for collecting information about the message to place in the provenance store, and uses the Loan Broker and CoTLooper workflows, giving a sampling of four different message types, three from Loan Broker and one from CoTLooper. Additionally, while CoTLooper is a toy workflow, the messages it passes utilizes real production APIs for Cursor on Target version 2 messages, serialized in XML. There is a significant impact for utilizing message reflection compared to no reflection[2]. The time to reflect within a message depends on how the class being

[1] All experiments were performed on a Linux 2.6.18 (CentOS 5.3) Quad-core box with 1.6Ghz processors and 4GB RAM, running MULE v2.1.2 and PLUS. All data is measured in nanoseconds, through the use of Java 1.6's System.nanoTime().

[2] Reflection in this case refers to calling a method dynamically at run-time, when the provenance capture mechanism was not compiled against the code defining that method.

reflected into is implemented; a simple "getter" method, such as getLatitude() from CoT Messages, which returns a stored value will be very fast. By contrast, a method which needs to parse a file will be much slower. The provenance capture mechanism cannot make any guarantee about how long it takes to invoke these methods, but it can provide infrastructure for doing so, and minimize the infrastructure cost.

4 Conclusions

In this work, we take the first step towards providing an automatic and simple mechanism for capturing provenance in open world systems. By enabling the MULE ESB with provenance collecting abilities, any application that is built to use MULE is automatically provenance enabled without underlying application modification or user knowledge. The approach captures previously implicit workflows, logging exactly what happened rather than what was expected to happen. In addition, no modifications to the capture mechanism are needed as applications evolve over time.

We see this work as an initial step toward multi-organizational provenance capture. Additional provenance collectors would be required, of course, since not all distributed, heterogeneous services use an ESB. We envision a variety of capture agents, each tailored to a different type of coordination point.

References

[1] Cursor on Target, `http://cot.mitre.org/`

[2] Altintas, I., Barney, O., Jaeger-Frank, E.: Provenance Collection Support in the Kepler Scientific Workflow System. In: Moreau, L., Foster, I. (eds.) IPAW 2006. LNCS, vol. 4145, pp. 118–132. Springer, Heidelberg (2006)

[3] Blaustein, B.T., Seligman, L., Morse, M., Allen, M.D., Rosenthal, A.: PLUS: Synthesizing privacy, lineage, uncertainty and security. In: ICDE Workshops, pp. 242–245 (2008)

[4] Buneman, P., Chapman, A., Cheney, J.: Provenance Management in Curated Databases. In: ACM SIGMOD, pp. 539–550 (2006)

[5] Frew, J., Metzger, D., Slaughter, P.: Automatic capture and reconstruction of computational provenance. Concurr. Comput.: Pract. Exper. 20, 485–496 (2008)

[6] Groth, P., Miles, S., Moreau, L.: PReServ: Provenance Recording for Services. UK OST e-Science second AHM (2005)

[7] Groth, P.T., Miles, S., Moreau, L.: A model of process documentation to determine provenance in mash-ups. ACM Trans. Internet Tech. 9 (2009)

[8] Missier, P., Belhajjame, K., Zhao, J., Goble, C.: Data lineage model for Taverna workflows with lightweight anotation requirements. In: Freire, J., Koop, D., Moreau, L. (eds.) IPAW 2008. LNCS, vol. 5272, pp. 17–30. Springer, Heidelberg (2008)

[9] Moreau, L., Ludäscher, B., et al.: Special Issue: The First Provenance Challenge. Concurrency and Computation: Practice and Experience 20, 409–418 (2008)

[10] Mulesoft.org, MULE 2.x (2009), `http://www.mulesoft.org/display/MULE2INTRO/Home`

[11] Muniswamy-Reddy, K.-K., Holland, D.A., Braun, U., Seltzer, M.I.: Provenance-Aware Storage Systems. In: USENIX, pp. 43–56 (2006)

[12] Scheidegger, C.E., Vo, H.T., Koop, D., Freire, J., Silva, C.: Querying and Re-Using Workflows with VisTrails. In: SIGMOD (2008)

[13] Simmhan, Y., Plale, B., Gannon, D.: Karma2: Provenance Management for Data Driven Workflows. Journal of Web Services Research 5 (2008)