Recording Provenance of Distributed Applications

Peter Buneman[1], Adrià Gascón[1], and Luc Moreau[2]

[1] School of Informatics, University of Edinburgh
[2] Electronics and Computer Science department, University of Southampton

We examine the following scenario, which we believe may be quite common. Two processes, possibly long-running, communicate with one another. Each keeps an incremental record of its provenance in PROV. What, in addition to the two provenance stores do we need to keep in order to record the communication and to allow a satisfactory account of the combined processes in PROV? An example[3] is of a research group communicating with a clinical data provider. Neither side – perhaps because of patient confidentiality and and confidentiality of research findings – wants to reveal their provenance to the other, but they want to record enough about the communication that a provenance record of the entire process could be constructed if needed – perhaps because the research had discovered the existence of a patient at risk.

Let us first examine what we want of the solution. Let us call the two processes, or parties, A and B. If A sends B some data we assume that A somehow has such data recorded as an entity d_1^A, similarly B records the received data as d_1^B. A subsequent piece of data d_2 from A to B will be recorded as d_2^A and d_2^B and so on. It is essential that we record the pairing $(d_1^A, d_1^B), (d_2^A, d_2^B), \ldots$. This pairing, or correspondence, might be recoverable from accurate timestamping, but even if this is available, it will not be enough if messages can be sent in parallel.

We believe the following are desirable.

1. We would like to record this pairing in PROV. The people who want to do this will already be using tools for recording PROV, and it should be possible to use PROV for reasoning about combined provenance.
2. The parties should disclose as little information as possible about their local provenance, in particular URI naming schemes should be kept private.
3. However, when A sends a message m to B, A should record least a URI that enables one to find the provenance graph of B, even if A is denied access to that graph. I.e., a third party examining A's provenance should find enough information to enable it to find the provenance graph of the recipient(s) of A's message m .
4. We want to use existing machinery – that belonging to A and B – for generating new URIs. We should not have to rely on a new "authority".
5. Information regarding attribution should be preserved in the combined provenance graph, e.g. it should be easy to check whether a given entity in the joint provenance graph was generated by A or B.
6. The combined provenance graph should support forward and backward reachability queries.

We believe the scenario is sufficiently important that there should be a standard method or framework for recording provenance of a decentralized process in such a way that a satisfactory account of the provenance for the whole process can be constructed from the provenance records of its components.

[3] Peter Buneman, Adrià Gascón and Dave Murray-Rust. Composition and Substitution in Provenance and Workflows, TAPP 2016.

In proposing a method it is important to take into account the sometimes conflicting requirements of confidentiality, traceability, responsibility and (bidirectional) reachability.

While the problem appears to be simple, the PROV family of specifications does not offer a solution for this. PROV-AQ only gives elements of solutions with the `has_provenance` header field and the provenance ping back. PROV bundles may also be part of the solution, but too many practical details about their use are left unspecified.