Architecture for Template-driven Provenance Recording

Elliot Fairweather, Pinar Alper, Talya Porat, and Vasa Curcin

King's College London
elliot.fairweather@kcl.ac.uk

Abstract. Provenance templates define abstract patterns of provenance data and have been shown to be useful when implementing support for provenance capture in existing software tools. Their strength is in exposing only the relevant provenance capture actions through a service interface, whilst hiding the complexities associated with managing the provenance data. We present an architecture for the creation and management of libraries of provenance documents constructed using templates.

Introduction

Provenance templates define abstract patterns of provenance data and have been shown to be useful when implementing support for provenance capture in existing software tools. Their strength is in exposing only the relevant provenance capture actions through a service interface, whilst hiding the complexities associated with managing the provenance data. We expand upon the formal model presented earlier in [1] by refining the methods by which provenance fragments generated by such templates are combined and integrated into an overall provenance document, and present an architecture for the creation and management of libraries of such documents constructed using templates.

Methodology

A provenance template [1] is a abstract fragment of a provenance document, that may be instantiated using concrete substitutions for variables contained with the template. Variables are of two kinds; identifier variables inhabiting the `var` namespace which are placeholders for node or relation identifiers, and value variables under `vvar`, which can be used in the place of an attribute value. A provenance template is itself a valid provenance document and as such allows nodes to be semantically annotated, allowing the inclusion of domain-specific information. Concrete provenance fragments are generated by an algorithm that accepts as input a template and substitution comprised of a set of variable-value bindings, and replaces variables for values in a copy of the template. We now present a system for constructing and managing PROV documents using templates.

Document types We distinguish three types of document, **target** documents, **template** documents and **fragment** documents. A target document is the document under construction. A system may manage the construction of multiple target documents at any given point. A template document describes a pattern representing a domain action to be replicated within a specific target document. A template document is registered to one particular target document. Call this target document the parent document of the template document. A fragment document is a document to be later merged into a target document, usually constructed by the instantiation of a template belonging to that document. A fragment document is also associated with a single target document, again referred to as its parent. Fragment documents perform a critical role in the document construction.

Metadata All documents are given a unique identifier and annotated with their type. This is stored separately as metadata and used to index documents in the management system. This metadata also includes which templates and fragments are associated to a target document and namespace data for each document. Metadata can be viewed as consisting of and represented by attribute-value pairs that belong to a document.

Namespace management A data document is created empty with a default namespace. Further namespaces may be added at any point. Except for the `ivar`, `vvar` and `pgt` namespaces, a template document may only include namespaces included in its parent document. A fragment document includes the `fragment` namespace used for auditing purposes and during the instantiation process. When a new fragment document is created it also inherits the namespaces of its parent document. New namespaces may not be added to fragment documents. This means that qualified names contained in substitutions given during instantiation must fall within the namespaces of the target document.

Fragment generation When a template is instantiated the graph generated is represented as a new fragment document. The generation of new fragments from templates may proceed in two ways, either **simultaneously** as a single step by applying a complete substitution, or **incrementally**, by first applying an initial substitution and then later applying zone substitutions. In both cases, it must be checked that the number of iterations of each zone falls within the bounds of that zone for the final instantiation to be valid. Simultaneous generation is defined algorithmically in Figure 6 of [1]. Incremental instantiation, however, in contrast to the description given there, is now considered to proceed in such a way that the fragment being generated is always connected. Entry and exit edges of zones are generated at the application of each zone substitution. In the case of serial zones, this requires that entry edges of the zone be repositioned upon each instantiation. Fragment documents are annotated with attribute-value pairs in the `fragment` namespace, as part of the mechanics of incremental instantiation but also for the purposes of auditing and analysis of the construction of the target document.

Merging and grafting When instantiation is finished, if the fragment document meets the iteration bound constraints for each zone the fragment is **merged** into the target document. This may result in the **grafting** of nodes. If the identifier of a node in a fragment pre-exists in the target document then that nodes is reused and a **graft** is created, joining the fragment and target documents. If the identifier of a fragment node does not exist in the target document a new node is created in the target. The merging process also adds additional attributes for the purpose of auditing. The validity of the target document is checked against the standard constraints of the PROV model following the merging of a fragment document. This is because the occurrence of grafting can lead to potential violations. If constraints are not met the merge is rolled back and the fragment removed from the target document.

Workflow We now proceed to give a detailed account of the workflow of the system. In a typical scenario, in order to construct a new document, a user would interact with the system in the following way.

1. Create a new target document Δ
2. Add necessary extra namespaces to target Δ
3. Register templates with target Δ
4. Create a new fragment document Φ belonging to target Δ
   (a) – by instantiating a template document with a complete substitution
   – i. by instantiating a template document T with an initial substitution
   ii. and then adding iterations to fragment Φ by instantiating zones of the template T with zone substitutions
   – by importing a standard PROV document
   (b) Merge fragment Φ into target Δ
5. Analyse and export target Δ or fragments of target Δ

Architecture

We now discuss the architecture of the proposed system with reference to the implementation of the first author. The overall structure of the architecture can be seen in Figure 1. The core of the system is the model component. Provenance documents are represented as graphs in which vertices and edges are typed and annotated with key-value pairs. The graph itself may also have key-value properties. Serialisation and deserialisation to PROV data formats is accomplished using the parsers provided by ProvToolbox library. Substitutions also form part of the model and parsers to both a proposed PROV-N format and JSON are given in the implementation. The template instantiation algorithm by which new fragment documents are generated from templates and substitutions is also defined within the model component. Storage of data in the system is abstracted by a persistence component to enable the use of different database technologies. By default, a Neo4j graph database is used but a relational database, SPARQL-enabled or alternative graph database could be used either instead or concurrently. The system is accessed via the document management component. This

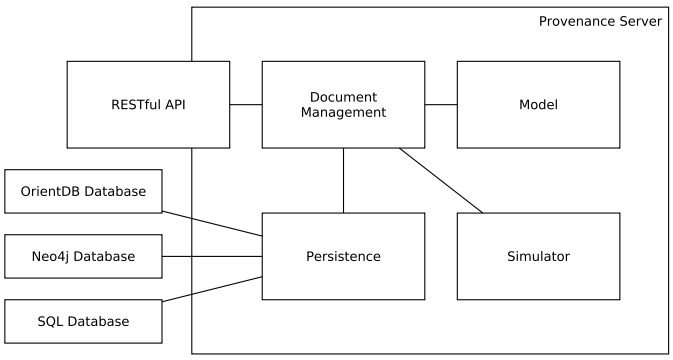

Fig. 1. Architecture

controls and executes operations outlined in the workflow, such as the creation of new target documents, namespace management, the registering of templates, and the generation and merging of new fragment documents. Fragment generation is achieved through interaction with the model component. Operations requiring the import, export or update of document data and metadata are supported via the persistence component. Access to the document management interface is provided via a RESTful web service. Documents and substitutions are passed to and from the server encoded as JSON and analysis is conducted by querying the underlying database. The specifics of a higher-level query interface for the system, agnostic to to a particular storage solution, is an area of ongoing research.

Conclusions and Future Work

This poster presented an architecture for capturing provenance data using templates. The design is intentionally generic, allowing a similar approach to be applied to any software architecture where it is preferable to capture provenance by mirroring actions from the main software system, rather than embedding it into a shared middleware. Ultimately, our goal is to facilitate the development of provenance back-ends and minimise the overheads involved in integrating provenance capture and utilisation into operational workflows. We have prototyped our architecture based on a decision aid software tool for communicating the risk of recurrent stroke to patients, that is being developed within the stroke theme of the Collaborative Leadership in Applied Healthcare Research and Care (CLAHRC) programme in South London. A key challenge for implementing provenance solutions is how to extract benefit from the captured data, and so, as the next step, we plan to devise user interface solutions for provenance reporting from decision support scenarios and utilise our group's previous experience in the area [2] to conduct a full quantitative and qualitative evaluation.

References

[1] V. Curcin et al. Templates as a method for implementing data provenance in decision support systems. *Journal of Biomedical Informatics*, 65:1–21, 1 2017.

[2] O. Kostopoulou et al. Diagnostic accuracy of GPs when using an early-intervention decision support system: a high-fidelity simulation. *British Journal of General Practice*, 2017.