WePIGE: The WebLab Provenance Information Generator and Explorer

Clément Caron
EADS-Cassidian, Val-de-Reuil
LIP6-UPMC, Paris

Bernd Amann
LIP6 - UPMC, Paris

Camelia Constantin
LIP6 - UPMC, Paris

Patrick Giroux
EADS-Cassidian, Val-de-Reuil

ABSTRACT

WePIGE illustrates a new approach for extracting fine-grained provenance information from XML artefact-based workflow executions. The extraction framework relies on the usage of XPath mapping rules for inferring data and service dependency links [2]. This demonstration illustrates the usage of the WePIGE graphical user interface for exploring the provenance graph generated by a predefined set of mapping rules and for semi-automatically inferring new provenance mapping rules between user selected XML fragments.

1. INTRODUCTION

Capturing and exploring fine-grained provenance information in complex data processing workflows is a challenging task which can be essentially divided into two approaches [8]. *Database provenance* is a "white-box" approach that consists of inferring fine-grained data provenance links from high-level (declarative) component specifications [3] or database queries [6, 11]. *Workflow provenance* is based on a "black-box" approach where dependency relationships are declared at run-time or are generated by using implicit data dependency patterns. The black-box setting has the advantage of not requiring any knowledge about the internal service behavior (code), but it generally also produces less precise provenance information with lower data granularity than more intrusive *white-box* solutions.

In [2] we have proposed a new non-intrusive declarative provenance model for XML data processing workflows. This model introduces *provenance mapping rules* "connecting" the results of two XPath expressions (extended by variables) evaluated on the XML input/output data of service calls. Defining such mapping rules requires the knowledge of XPath and might be a difficult task. The model proposed is quite similar and partly inspired of [5], the main difference remains in the use of the WebLab platform, and its insertion-only semantic (data is never modified nor deleted). The WePIGE prototype presented here intends to ease this specification task by guiding the *workflow designer* through a graphical interface that allows him to add new mappings, visualize their effect on some XML documents and possibly adapt them manually in order to be more precise. The resulting data dependencies can be further explored by *workflow users* (possibly the same as the designers) in order to detect possible dysfunctional services or erroneous data which led to unexpected execution results.

Some existing works on scientific workflow provenance, such as *Taverna* [9], *Kepler* [1] or *Vistrails* [7] also use the notion of *provenance mappings* for inferring data and service dependencies. The main differences with our approach is that in these systems, mappings are part of the *workflow* description and take account of the control flow. In our setting, mapping rules are defined for each service independently of a given workflow (control-flow) and can be reused in different contexts. This preserves the notion of service reusability of the *Service Oriented Architecture* framework. A second important feature of our artefact-based workflow and provenance model is the possibility to generate provenance information *online* (during the workflow execution) and *offline* (after the workflow execution).

In the following, we will demonstrate the usage of WePIGE for accomplishing two separate tasks : (*i*) the assisted design of provenance mapping rules and (*ii*) the exploration of provenance information generated by these rules.

The rest of the article is organized as follows. We will present our provenance generation model in Section 2. Section 3 explains the overall architecture of our framework. The WePIGE user interface and mapping generation algorithm are described in Section 4 and the article finishes with two demonstration scenarios in Section 5.

2. WEBLAB PROVENANCE MODEL

Within the WebLab platform (see Section 3), data used and generated by a service workflow are collected within a single XML document validated by a predefined DTD. Each service call receives as input such a *WebLab* document and extends it with new XML fragments. This "append" semantic guarantees that no data is ever deleted and the WebLab documents contains the final output but also all intermediate results produced by each workflow step.

Consider the WebLab document in Figure 1 generated by three subsequent service calls *Normalizer* at time instant t_1, *LanguageExtractor* at time instant t_2 and *Translator* at time instant t_3. All nodes in the document are labeled by their XML element name and the creation time-stamp (we assume that each time-stamp identifies a unique service call). Nodes that existed before the first service call are labeled by time instant t_0.

In order to infer fine-grained provenance links between the input and the output of services, the workflow designer

EDBT/ICDT '14 March 24 - 28 2014, Athens, Greece

10.5441/002/edbt.2014.69

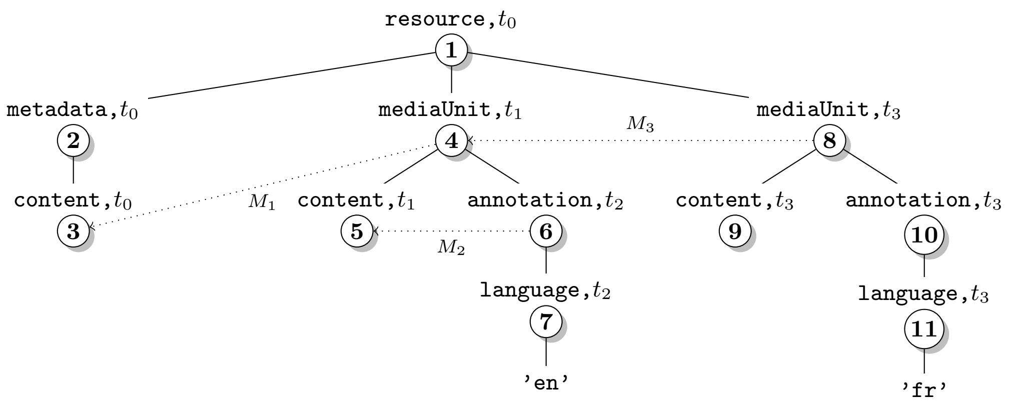


Figure 1: WebLab Document

has to specify *provenance mapping rules* that are based on *XPath patterns extended with variables* which will be applied on the final document produced by the workflow execution (for more details about the mapping model and the dependency graph generation, see [2]). For example, service *Normalizer* transforms *content* resource ③ and produces a normalized version, *mediaUnit* resource ④, which is added as a child of the initial *resource* node ①. This input-output data dependency of service *Normalizer* can be specified by the following simple mapping rule:

$$M_1 : \frac{/resource/metadata/content}{/resource/mediaUnit}$$

Observe that the provenance mapping semantics takes account of the temporal dependencies between XML fragments (a fragment can only depend on previously generated fragments) [2]. Similarly, service *LanguageExtractor* identifies the language of the normalized *content* element of some *mediaUnit* and adds the result as a child of type *annotation*:

$$M_2 : \frac{//mediaUnit[\$x := @uri]/content}{//mediaUnit[\$x := @uri]/annotation[//language]}$$

Defining such mapping rules is not easy and requires knowledge of the XPath syntax and semantics. It is also obvious that the expressivity of this rule language depends on the expressivity of XPath and the underlying document DTD. A particular challenge concerns workflows where the same service can be called several times which makes the exact characterization of the input corresponding to the output of a specific service call difficult (each call generates data of the same type). The formal study of this expressivity remains an open question. However, we believe that our mapping language is sufficient for many real-world workflows and a first step towards a high-level provenance information generation language.

3. ARCHITECTURE

We have integrated this provenance model into *WebLab*, an open environment for composing software components into complex media mining workflows. Following the core architecture of the platform, all components are implemented in Java and deployed within the service-oriented processing middle-ware PEtALS[1]. The workflow execution engine is based on XML for representing data and on RDF for encoding meta-data and ontologies.

The overall architecture is shown in Figure 2. It is composed of three separate systems. The **WebLab Platform** is responsible for the execution of the workflows. The *Service Catalog* contains meta-data about services including the service endpoints and signatures. Each *Workflow execution* generates a single XML document in the XQuery-enabled *WebLab repository*. All XML fragments generated by a given service call are identified in the repository and stored by the *Recorder* with all generated meta-data (*workflow execution identifier*, *service identifier*, *timestamp*) in the *Execution Trace* triple-store for future use by the WebLab PROV system. The **WebLab PROV** system [2] generates and stores

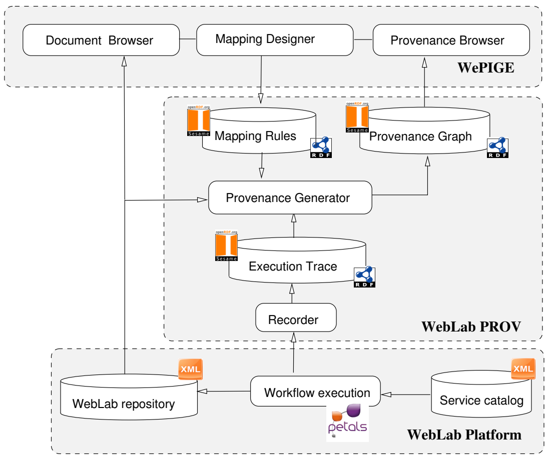


Figure 2: Architecture of WebLab PROV

the provenance graphs of service executions. The mapping rules are stored in the *Mapping Rules* RDF repository. The *Provenance Generator* represents the heart of the system. It applies mapping rules to WebLab documents by combining them with the corresponding execution trace information. The generated data and service dependencies are stored in the *Provenance Graph* RDF repository. Provenance infor-

[1]http://petals.ow2.org/

mation is stored according to the RDF-PROV ontology [4] and using an RDF triple-store allows us to use SPARQL endpoints for querying generated provenance graphs[2]. The **WePIGE** system allows users to interact with the WebLab PROV system to explore WebLab document trees and the data dependency graphs generated by a set of provenance mappings. The main component of WePIGE is the *Mapping Designer*, which helps the user to create provenance mapping rules for services.

4. WePIGE USER INTERFACE

The WePIGE user interface can be used in two different modes: (*i*) the Provenance Browser mode, that assists users in exploring provenance links to detect possible data or execution errors and (*ii*) the Mapping Designer mode, that visually assists workflow designers in defining provenance mapping rules. The general user interface is illustrated in

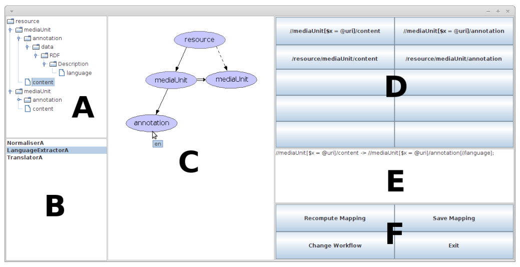


Figure 3: WePIGE User Interface Layout

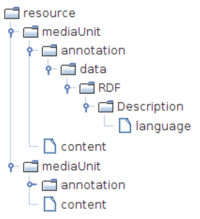
Figure 3. It is divided into six separate frames identified by letters **A** to **F**. Frame **F** is available in both modes and contains control buttons. In the following, we will describe in more detail the other frames and their usage in both interface modes.

4.1 Provenance Browser mode

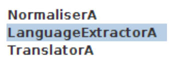
The graphical user interface (GUI) in the Provenance Browser mode assists the *workflow user* in exploring the provenance graph generated by some predefined mapping rules on a specific WebLab document. The expert first will choose an execution identifier and then browse the corresponding document and the workflow execution.

Frame **A** (Figure 4(a)) contains the *Document Browser*. At initialization, it shows the document tree structure of the final document generated by the selected workflow execution. Frame **B** (Figure 4(b)) contains the list of all service calls of the chosen workflow execution. Finally, frame **C** displays the provenance graph generated by a set of predefined provenance mapping rules displayed in Frame **E** (see Section 4.2 for the generation of these rules). Frame **C** uses different arrow types depending on their semantics: dashed arrows for hierarchical XML child relations, double arrows

[2]Since most of the other WebLab components are compliant with the *W3C* standards (XML, RDF), we chose the W3C recommendation *PROV-O* as provenance ontology instead of other approaches like the Open Provenance Model (OPM, http://openprovenance.org/). OPM was developed as a generic provenance model in the context of workflow provenance, while PROV-O includes a standard mapping to RDF and can directly be used in the WebLab platform which follows a semantic web approach.

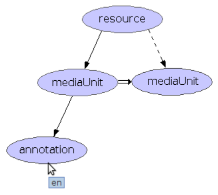


(a) Frame A

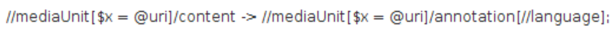


NormaliserA
LanguageExtractorA
TranslatorA

(b) Frame B



(c) Frame C



//mediaUnit[$x = @uri]/content -> //mediaUnit[$x = @uri]/annotation[//language];

(d) Frame E

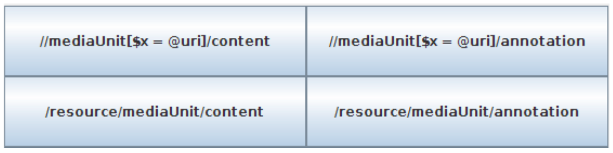
Figure 4: Zoom on frames A and C

for provenance links, and simple arrow for reflecting both, hierarchical and provenance, semantics.

In order to track down a problem within a workflow execution, users can browse within each of these three frames by clicking on any document node and/or service call. For example, suppose the user selected a call to service *LanguageExtractor* which extracted a language `annotation` node from a `mediaunit` resource. Frame **C** then automatically shows the data provenance sub-graph (Figure 4(c)) whereas frame **A** displays the document state before the service call. All input resources of the service call are highlighted in frame **A**, whereas the newly created resources are colored in red in the provenance graph. This automatic synchronization between frames allows the expert to explore step by step the data generation process by selecting service executions and document nodes.

4.2 Mapping Designer Mode

This mode assists the workflow designer to generate mapping rules semi-automatically. The main difference between the provenance browsing mode and the mapping designer mode concerns the semantics of user actions on frame **A** and the interaction with frame **D** (Figure 5(a)) which displays the data dependency rules generated by the mapping generation algorithm described below.



//mediaUnit[$x = @uri]/content	//mediaUnit[$x = @uri]/annotation
/resource/mediaUnit/content	/resource/mediaUnit/annotation

(a) Frame D

Figure 5: Frame D

The user will first choose in frame **B** a service call he wants to define a mapping rule for. As in the provenance browser mode, frame **A** displays the document state which existed just before the service call (this information can be inferred thanks to the time-stamps generated by the *Recorder* during the workflow execution), and highlights all nodes detected as input with the current data dependency rule. Frame **C** displays the provenance graph generated by the default rule $//* \rightarrow //*$ (frame **E**) which links all input resources to all new resources created by the service call. The user can then refine the default rule by choosing a set of input fragments I in frame **A** and a set of output fragments O in frame

C. The mapping generation algorithm presented below produces a set of candidate mappings displayed in frame **D** (this part is empty in the browsing mode). The user can choose among these candidates one mapping which is then displayed in frame **E**. Using this frame, the expert can also modify XPath mapping rules manually if needed. At each moment, the interface provides feedback to the user by continuously synchronizing the provenance graph displayed in frame **C** according to the mapping rule in frame **E**.

Mapping Generation;.

The main goal of the mapping generation algorithm is to infer mapping rules from a set of user selected input fragments I to a set of user selected output fragments O of some document d. We implemented two algorithms.

The first algorithm is based on the query learning algorithm work described in [10]. This algorithm computes for both sets I and O the most precise common XPath expressions $xp(I)$ and $xp(O)$ such that $I \subseteq xp(I)(d)$ and $O \subseteq xp(O)(d)$. The resulting mapping rule is:

$$xp(I) \to xp(O)$$

The second algorithm is an extension of the first algorithm by adding variables shared between the input and output XPath expressions. This is based on the simple heuristic that service calls generally extend the XML document locally with respect to some specific XML fragment. The algorithm computes the nearest common ancestor a of all nodes in I and O. If a is different from the document root, there exists a common prefix $xp(a)$ of $xp(I)$ and $xp(O)$ such that $xp(I) = xp(a)/xp_1$ and $xp(O) = xp(a)/xp_2$. We can then define a new mapping rule:

$$xp(a)[\$x = @uri]/xp_1 \to xp(a)[\$x = @uri]/xp_2$$

Both results are displayed for the users, as explained in the next section.

5. DEMONSTRATION SCENARIO

During the demonstration, we plan to make a full presentation of the prototype using different datasets created from different media mining projects.

The presentation will be divided into three main parts. At first, we will get familiar with the GUI by browsing a simple resource (around 20 identified nodes and a dozen provenance links) created by a previously setup workflow from the WebLab platform, composed of fully functional rules. This resource will contain a text and its translation, as well as several segments corresponding to the keywords extracted with a *named entity extractor*. During this part, we will see how a user can apply data dependency rules on services, explore data and service calls through their provenance links and detect possible problems within an execution.

In the second part of the demonstration, we will show how a user can add a service in the previous workflow, and how the application will help him to write new rules.

In the last part, we will show how the application reacts during the generation of large provenance graphs for large XML files. We will use data produced by executions of actual WebLab workflows extracting information from videos within the AXES[3] project. These workflows generate thousands of identified XML resources, where each resource is composed of multiple annotations and video segments, including links to the *normalised* version of the video, the *transcription*, the *extraction* of key shots, and more. We will generate a dozen of workflow executions in advance and apply provenance rules which will generate thousands of provenance links for each workflow.

[3]FP7 programme AXES ICT-269980

6. CONCLUSION

In our future work we plan to increase the expressiveness of our provenance model by extending provenance rules with negation and by adding a (meta-)provenance layer annotating each provenance link with its generation rule(s). We are also currently working on the extension of our provenance model with a quality inference layer for analyzing and improving the results of complex text mining workflows.

7. REFERENCES

[1] I. Altintas, C. Berkley, E. Jaeger, M. Jones, B. Ludäscher, and S. Mock. Kepler: An extensible system for design and execution of scientific workflows. In *Proc. SSDBM*, pages 423–424, Washington, DC, USA, 2004.

[2] B. Amann, C. Constantin, C. Caron, and P. Giroux. Weblab prov: computing fine-grained provenance links for xml artifacts. In *Proc. of the Joint EDBT/ICDT Workshops*, pages 298–306. ACM, 2013.

[3] Y. Amsterdamer, S. B. Davidson, D. Deutch, T. Milo, J. Stoyanovich, and V. Tannen. Putting lipstick on pig: enabling database-style workflow provenance. *Proc. VLDB Endow.*, 5(4):346–357, Dec. 2011.

[4] K. Belhajjame, J. Cheney, D. Corsar, D. Garijo, S. Soiland-Reyes, S. Zednik, and J. Zhao. PROV-O: The PROV ontology. Technical report, 2012.

[5] S. Bowers, T. McPhillips, and B. Ludäscher. Declarative rules for inferring fine-grained data provenance from scientific workflow execution traces. In *Proc. of the 4th Int. Conf. on Provenance and Annotation of Data and Processes*, pages 82–96, Berlin, Heidelberg, 2012.

[6] P. Buneman, J. Cheney, and S. Vansummeren. On the expressiveness of implicit provenance in query and update languages. *ACM Trans. Database Syst.*, 33(4):28:1–28:47, Dec. 2008.

[7] S. P. Callahan, J. Freire, E. Santos, C. E. Scheidegger, C. T. Silva, and H. T. Vo. Vistrails: visualization meets data management. In *Proc. of the ACM SIGMOD Int. Conf. on Management of Data*, pages 745–747, New York, NY, USA, 2006.

[8] F. Chirigati and J. Freire. Towards integrating workflow and database provenance. In *Proc. of the 4th Int. Conf. on Provenance and Annotation of Data and Processes*, pages 11–23, Berlin, Heidelberg, 2012.

[9] D. Hull, K. Wolstencroft, R. Stevens, C. Goble, M. R. Pocock, P. Li, and T. Oinn. Taverna: a tool for building and running workflows of services. *Nucleic acids research*, 34:W729–W732, July 2006.

[10] S. Staworko and P. Wieczorek. Learning twig and path queries. In *Proc. of ICDT*, pages 140–154. ACM, 2012.

[11] Y. Theoharis, I. Fundulaki, G. Karvounarakis, and V. Christophides. On provenance of queries on semantic web data. *IEEE Internet Computing*, 15(1):31–39, Jan. 2011.