A Canonical Form for PROV Documents and its Application to Equality, Signature, and Validation

Luc Moreau, University of Southampton

We present a canonical form for PROV that is a normalized way of representing PROV documents as mathematical expressions. As opposed to the normal form specified by the PROV-CONSTRAINTS recommendation, the canonical form we present is defined for all PROV documents, irrespective of their validity, and it can be serialized in a unique way. The article makes the case for a canonical form for PROV and its potential uses, namely: comparison of PROV documents in different formats, validation, and signature of PROV documents. A signature of a PROV document allows the integrity and the author of provenance to be ascertained; since the signature is based on the canonical form, these checks are not tied to a particular encoding, but can be performed on any representation of PROV.

CCS Concepts: •**Information systems** → **Semantic web description languages; Data encoding and canonicalization;** •**Security and privacy** → **Digital signatures;** •**Theory of computation** → **Data provenance;**

1. INTRODUCTION

Provenance is adopted in an increasing number of applications with a view to provide novel functionality. For instance, in climate science[1], it is used to link up all relevant scientific artefacts leading to some conclusions; in the legal context[2]; it provides details of the minute steps involved in transforming legal notices before publication; in the "Araport" biology portal[3], it is used to provide credits to datasets and services publishers; in disaster response [Ramchurn et al. 2015], it helps provide notifications when knowledge of the situation invalidates previous plans.

The World Wide Web Consortium (W3C) defines provenance as a *record that describes how entities, activities, and agents have influenced a piece of data* [Moreau and Missier (eds.) 2013]. Such a record is only useful if it can be trusted. To ascertain trust of a provenance record, the W3C further provides a mechanism to express the provenance of provenance, allowing the context of the creation of a provenance record to be described; it can include information such as the provenance author, its creation date, and its generation process. Again, such a record is useful as long as it can itself be trusted to be an authentic description. Thus, some have begun to argue for the use of signatures for provenance records [Hasan et al. 2009; Gadelha and Mattoso 2008; Seneviratne and Beckett 2016], while others have simply implemented their own method of signing provenance[2]. A cryptographic signature of provenance guarantees its integrity (meaning that it has not been tampered with) and its non-repudiation (meaning that the signer cannot deny its existence) [van Tilborg 2005].

[1] http://nca2014.globalchange.gov/report
[2] https://www.thegazette.co.uk/
[3] https://www.araport.org/

This work is funded in part by the EPSRC SOCIAM (EP/J017728/1) and ORCHID (EP/I011587/1) projects, the FP7 SmartSociety (600854) project, and the ESRC eBook (ES/K007246/1) project.
Author's addresses: Luc Moreau, Web and Internet Science Group, Electronics and Computer Science, University of Southampton, Southampton SO17 1BJ.
Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored. For all other uses, contact the owner/author(s).
© 2017 Copyright held by the owner/author(s). 1533-5399/2017/00-ARTxxxx $15.00
DOI: http://dx.doi.org/10.1145/3032990

Against this background, PROV is a set of W3C specifications [Groth and Moreau (eds.) 2013] aiming to facilitate the representation and exchange of provenance on the Web. PROV consists of a conceptual data model and serializations to various technologies. The support of multiple technologies is exploited by some provenance-enabled applications that use pipelines to process multiple PROV representations: a pipeline may, for instance, capture provenance with PROV-XML [Hua et al. 2013] or PROV-N [Moreau et al. 2013], store it in RDF (PROV-O [Lebo et al. 2013]), query it with SPARQL, and perform client-side rendering using PROV-JSON [Huynh et al. 2013] retrieved by content negotiation.

Therefore, at any point in the pipeline, a component may have to sign or verify a signature of the provenance it processes. Thus, there is a requirement for signatures to be defined on a logical form of PROV. A similar requirement has been tackled in different contexts, by developing a logical representation of some data model, before signing a serialization into an array of bytes [Eastlake and Jones 2001]: XML Signature [Eastlake et al. 2008] defines Canonical XML, a serialization of XML invariant to some permitted syntactic changes; RDF signature is based on a canonical form of RDF [Carroll 2003], similarly exploited to sign JSON-LD messages [Sporny 2015; Longley and Sporny 2016]. The canonization process of PROV differs from the canonization of an XML or RDF representation of PROV, because there are logically equivalent PROV expressions that do not have a unique mapping to XML or RDF.

The problem of signature for PROV documents is further compounded because the W3C did not define equivalence between the various PROV serializations. If a canonical representation of PROV was defined, equality of two representations r_1 and r_2 could be defined in terms of equality of their respective canonical representations c_1 and c_2. Only then, would it become possible to obtain a signature for representation r by signing its canonical representation c.

The problem of defining a canonical form for PROV has been addressed in part. The PROV-CONSTRAINTS specification [Cheney et al. 2013] defines a notion of normal form, as a necessary intermediate representation before checking whether a PROV document is consistent, i.e. whether it has a plausible logical interpretation. Unfortunately, not all PROV documents have a normal form. Indeed, according to PROV-CONSTRAINTS, there is a set of invalid provenance documents that do not have a normal form. It means that we could not sign these nor determine their equivalence. So, a notion of normal form for PROV that is defined independently of PROV validity is a critical step towards PROV signature and serialization equivalence.

Thus, the contributions of this paper are as follows: i) a canonical form for PROV; ii) a notion of equality for PROV representations; iii) a signature technique based on PROV Canonical Form; iv) a reformulation of the validation algorithm in terms of PROV Canonical Form; v) an empirical evaluation demonstrating the cost of transforming to canonical form and producing signatures.

The rest of the paper is organized as follows. Section 2 provides a more detailed discussion on why a canonical form of PROV is needed. Section 3 introduces a few preliminaries, before Section 4 defines PROV Canonical Form and a function to convert any document into this canonical form. Section 5 then discusses three applications of PROV Canonical Form, namely conversion between serializations, signatures, and validation. Section 6 embarks on an empirical evaluation of the conversion to canonical form. Section 7 contrasts this approach to related work. The paper then concludes with a discussion of future extensions in Section 8.

2. THE CASE FOR A CANONICAL FORM FOR PROV

The PROV data model is a vocabulary to express the provenance of resources. It consists of three core classes: An *entity* is a piece of data, a data set, a document, a decision we want to describe the provenance of. An *activity* is something that happened in the world. An *agent* is responsible for activities, entities and other agents. Those concepts can be linked with some associations. Two such associations are as follows. An entity may be *generated by* an activity: for instance, a picture was the result of a drawing activity. An entity may be *derived from* another entity: a plot was calculated from a data set. We refer the reader to the definition of PROV for a complete description of the vocabulary [Moreau and Missier (eds.) 2013].

2.1. Transformation Derived from PROV Constraints

Let us consider a PROV document D1 consisting of a single triple stating that a resource e was generated by resource a:

```
:e prov:wasGeneratedBy :a.                                                   (D1)
```

According to PROV-CONSTRAINTS [Cheney et al. 2013], Constraint PCO50 (typing) allows us to determine that e is of type “entity”, whereas a is of type “activity”. An alternate document D2 could be constructed including this typing information.

```
:e a prov:Entity.
:a a prov:Activity.                                                          (D2)
:e prov:wasGeneratedBy :a.
```

Obviously, documents D1 and D2 are syntactically different, since they contain a different number of triples, but they are logically equivalent. Determining the domain and range of properties can be supported by RDFS-reasoning [Brickley and Guha 2014]. However, PROV comes with further reasoning rules.

Let us consider the following document D3. The first “block of triples” is a description that (entity) e was generated by (activity) a at some point in time. The second block of triples is a description that the same entity was generated by the same activity at some location.

```
:e prov:qualifiedGeneration [prov:activity :a;
                             prov:atTime "2012-03-31T09:21:00"^^xsd:dateTime].        (D3)

:e prov:qualifiedGeneration [prov:activity :a;
                             prov:atLocation dbpedia:Southampton].
```

According to PROV-CONSTRAINTS [Cheney et al. 2013], Constraint PCO24 (unique-generation) requires the unicity of generation for a given entity-activity pair. Thus, we infer the existence of a single generation that took place at the time and location indicated, which can be expressed in document D4 as follows.

```
:e prov:qualifiedGeneration [prov:activity :a;
                             prov:atTime "2012-03-31T09:21:00"^^xsd:dateTime;         (D4)
                             prov:atLocation dbpedia:Southampton].
```

Again, documents D3 and D4 are distinct but logically equivalent according to PROV.

These two examples illustrate that within a given representation, here RDF, we can construct documents that are distinct according to RDF model theory [Hayes and Patel-Schneider 2014], but are logically equivalent according to PROV. To be able to establish semantic equivalence of documents, it is, therefore, necessary to convert them into a canonical form, which can then be comparable with syntactic means.

2.2. Round-Trip Conversion Between Representations

Libraries, such as ProvToolbox[4], are capable of converting between PROV representations, by reading a given format, building an internal representation (here in Java),

[4]http://lucmoreau.github.io/ProvToolbox/

and then converting it to other representations. Ideally, round-trip conversions, say from PROV-N to RDF and back, should "preserve" the original representation. The core of the PROV data model is easily mapped to all serializations. For instance, `e prov:wasGeneratedBy a` in RDF is simply mapped to `wasGeneratedBy(e,a,-)` in PROV-N, and vice-versa. For those cases, round-trip conversions are easily proven correct.

However, there are many cases, for which the PROV family of specifications does not provide guidance on how to ensure safe round-trip conversion. For instance, document D5 contains two descriptions about a single generation event (named gen1), respectively involving two entities e1 and e2, and two different times.

```
wasGeneratedBy(gen1; e1, a, 2012-03-31T09:21:00)                        (D5)
wasGeneratedBy(gen1; e2, a, 2012-03-31T09:22:00)
```

Conversion to RDF leads to D6:

```
:e1 prov:qualifiedGeneration :gen1.
:e2 prov:qualifiedGeneration :gen1.                                     (D6)
:gen1 prov:activity :a.
:gen1 prov:atTime "2012-03-31T09:21:00"^^xsd:dateTime.
:gen1 prov:atTime "2012-03-31T09:22:00"^^xsd:dateTime.
```

Conversion back to PROV-N *could* result in document D7 in which it appears that the generation times have been swapped, compared to D5.

```
wasGeneratedBy(gen1; e1, a, 2012-03-31T09:22:00)                        (D7)
wasGeneratedBy(gen1; e2, a, 2012-03-31T09:21:00)
```

This example is challenging because it does not have a normal form according to PROV-CONSTRAINTS, since gen1 is a key for the two expressions, and thus, parameters in corresponding positions (e1 and e2) are expected to be the same. Therefore, in absence of PROV-CONSTRAINTS normal form, we are unable to compare D5 and D7. PROV-CONSTRAINTS declares them invalid. Thus, we argue that the validity of a document should not affect our ability to convert it to other representations, to provide correct round-trip conversion for it, and to define its canonical representation.

2.3. Signing Documents

Whether a provenance document is valid or not, it is essential to be able to sign it, since a digital signature allows us to authenticate it (determine its author) and ascertain its integrity (determine whether it has been altered).

ProvStore [Huynh and Moreau 2014] is an example of a provenance management system that accepts provenance in any format, stores it in a SQL database, and allows for it to be retrieved in any PROV-supported format. A consumer of provenance may want to validate a signature of a provenance document, irrespective of the format in which it was being deposited and is being accessed. A canonical representation of PROV documents is hence required to be able to sign such documents in any representation.

2.4. Requirements for PROV Canonical Form

From the discussion in the previous sections, we derive the following requirements to be satisfied by PROV Canonical Form. We discuss them below.

REQUIREMENT 2.1 (UNIVERSAL CANONICAL FORM). PROV *Canonical Form must exist for all syntactically correct* PROV *constructs, whether valid or not.*

REQUIREMENT 2.2 (INVARIANCE). PROV *Canonical Form is invariant to statement order and* PROV *inferences in* PROV-CONSTRAINTS *[Cheney et al. 2013].*

REQUIREMENT 2.3 (CANONICAL FORM EQUALITY). *Equality of terms in* PROV *Canonical Form must be defined.*

REQUIREMENT 2.4 (CLOSED UNDER MERGE). *A* PROV *Canonical Form must exist for any term resulting from the merging of two terms in* PROV *Canonical Form.*

REQUIREMENT 2.5 (CANONICAL FORM DISTINGUISHED SERIALIZATION). PROV *Canonical Form must have a unique distinguished serialization suitable for signing.*

There is an expectation of universality, guaranteeing that all PROV documents can be given a canonical form (Requirement 2.1). Provenance documents in a given format typically have multiple representations because of different statement order: these variants should have a single canonical form, but also applying common PROV inferences should not alter a document's canonical form (Requirement 2.2). We aim to define equality of representations in terms of the equality of their canonical forms, so the latter needs to be defined (Requirement 2.3).

In systems running continuously, one may want to understand how today's behavior differs from the past week's behavior. Thus, in such a context, incremental processing of provenance is critical. A fundamental operation that must be supported is the ability to merge provenance. While it is understood that validity is not preserved by the merge operation [Cheney et al. 2013; Kwasnikowska et al. 2015], it is critical that one can derive a canonical form from the merge of two canonical forms (See Requirement 2.4). To be able to sign a document, there needs to be a unique serialization of its canonical form (Requirement 2.5).

The normal form (NF) defined by PROV-CONSTRAINTS satisfies Requirements 2.2 and 2.3. However, the process of transforming to NF is a partial function that is not defined when unification fails; thus, NF does not meet Requirement 2.1. Combining two documents in NF may result in a document for which there is no NF, thus failing Requirement 2.4. PROV-CONSTRAINTS does not define a serialization format for NF, in particular given the introduction of existential identifiers which require similar treatment to RDF blank nodes; hence, Requirement 2.5 is not met for NF. Given this, there is a need for a novel PROV Canonical Form, which addresses all these requirements. We present it in Section 4, after some preliminaries.

3. PRELIMINARIES

In the following section, we define a function to transform PROV documents into their canonical form. We adopt a functional style and rely on a few common combinators of functional programming [Odersky et al. 2010]. We summarise them below and also provide a link to their Scala definition.

$map(S, f)$	Builds a new set by applying a function f to all elements of set S (see Scala doc)
$fmap(S, f)$	Builds a new set by applying a function f to all elements of set S and taking the union of the resulting sets (see Scala doc)
$groupBy(S, f)$	Partitions set S into a map of sets according to some discriminator function f (see Scala doc)
$reduce(S, f)$	Reduces the elements of set S using the associative binary operator f (see Scala doc)
$M.values$	Collects all values of a map M in a set (see Scala doc)
$mapValues(M, f)$	Transforms map M by applying a function f to every retrieved value (see Scala doc)

4. PROV CANONICAL FORM

4.1. Idealized PROV Language

In this section, we present an idealized definition of PROV, for which we subsequently investigate a canonical form. This version is idealized because it aims to minimize the number of terms that need to be handled. We make it explicit where we deviate from the normative definition of PROV [Moreau and Missier (eds.) 2013]. We adopt a language syntax that is closely inspired by the functional notation PROV-N [Moreau et al. 2013]. We limit our presentation to the syntactic dimension of PROV; for semantic aspects, we refer the reader to normative documents [Moreau and Missier (eds.) 2013; Cheney et al. 2013].

According to Figure 1, a PROV document is a set of bundles and terms. A bundle is a named set of terms. Bundles were introduced in PROV to be able to express the provenance of provenance. By definition, the contents of documents and bundles have an unspecified order.

Terms in Idealized PROV can be nodes ($term_1$) or relations ($term_2, \ldots, term_4$).

A bundle or a node must be identified by a name (in practice, a URI). An instance of a relation connecting several nodes can optionally be identified by a name as well. This is marked by the notation (η_o).

All terms, whether nodes or relations, can contain further descriptions in the form of key-value pairs. Keys are of the form (κ), whereas values can be names (η) or primitive constants (π), such as numbers, booleans, or strings.

Relations are primarily binary relations, linking an *influencee* η_e to an *influencer* η_r, always appearing right after the optional identifier.

In accordance to PROV, we distinguish three categories of relations. Relations denoted by $term_2$ are pure binary relations between η_e and η_r with an optional identifier η_o. Relations denoted by $term_3$ are ternary relations, which associate an influencer η_r and an influencee η_e with a further node η_1. Relations are further subtyped and Figure 1 displays the permitted predicates for each relation. Finally, relations denoted by $term_4$ make associations between five names and are also optionally identified with η_o; the five names include an influencer η_r and an influencee η_e, but also a further node η_1, and two further identifiers η_2, η_3 intended to denote two relations; this construct provides the means to express the details of a derivation (see `prov:Derivation` [Moreau and Missier (eds.) 2013]).

Constructs

$$\begin{array}{rcl}
document & ::= & \textsf{document}\ (bundle \mid term)^* \\
bundle & ::= & \textsf{bundle}(\eta_i, term^*) \\
term & ::= & term_1 \mid term_2 \mid term_3 \mid term_4 \\
term_1 & ::= & node(\eta_i, [\kappa_i = \eta, \ldots, \kappa_j = \pi, \ldots]) \\
term_2 & ::= & rel_1(\eta_o, \eta_e, \eta_r, [\kappa_i = \eta, \ldots, \kappa_j = \pi, \ldots]) \\
term_3 & ::= & rel_2(\eta_o, \eta_e, \eta_r, \eta_1, [\kappa_i = \eta, \ldots, \kappa_j = \pi, \ldots]) \\
term_4 & ::= & rel_3(\eta_o, \eta_e, \eta_r, \eta_1, \eta_2, \eta_3, [\kappa_i = \eta, \ldots, \kappa_j = \pi, \ldots]) \\
document & \in & Document \\
bundle & \in & Bundle \\
term & \in & Term
\end{array}$$

PROV Predicates

$node \in \{$entity, activity, agent$\}$

$rel_1 \in \{$wasGeneratedBy, used, wasAttributedTo, wasInvalidatedBy, wasInformedBy, wasInfluencedBy, specializationOf, alternateOf, hadMember$\}$

$rel_2 \in \{$wasStartedBy, wasEndedBy, wasAssociatedWith, actedOnBehalfOf$\}$

$rel_3 \in \{$wasDerivedFrom$\}$

Sets

$\eta \in \mathcal{N}$	Set of names
$\pi \in \mathcal{P}$	Set of constants, with total order $\pi_i < \pi_j$ if $i < j$
$\kappa \in \mathcal{K}$	Set of keys, with total order $\kappa_i < \kappa_j$ if $i < j$

Positional roles

η_e	influence$\underline{e}$
η_r	influence$\underline{r}$
η_i	mandatory $\underline{i}$dentifier
η_o	$\underline{o}$ptional identifier
$\eta_{1,2,3}$	optional argument

Fig. 1. Idealized PROV

Idealized PROV deviates from normative PROV in two ways. Some PROV binary relations of type specializationOf, alternateOf, hadMember do not allow for attributes and optional identifiers [Moreau 2016]: thus, Idealized PROV is a superset of normative PROV. We have added these features for reasons of uniformity and for addressing some use cases [Moreau 2016]. The PROV-N notation allows for time annotations in various positions of the model. These are not explicit arguments of the Idealized PROV functional notation, but instead, should be encoded as part of the key-value pairs. So, overall, all terms of normative PROV can be encoded in Idealized PROV.

4.2. Mergeable PROV

To support incremental processing of provenance, we justified Requirement 2.4 by the need to merge provenance. Thus, we introduce the notion of mergeable PROV, displayed in Figure 2. The structure of documents is similar to that of Idealized PROV (Figure 1), except for names being replaced by sets of names. The rationale for this design stems from constraints defined in PROV-CONSTRAINTS, which we now explain.

According to PCO23 (key-properties), the identifier field η_0, if defined, is a key for a relation; thus, it uniquely determines the other parameters of that relation. As we are not trying to determine whether constraints are satisfied, we do not seek to unify identifiers as in PROV-CONSTRAINTS; instead, inspired by this constraint, we allow for relations with the same identifier to be merged, by taking the union of their parameters in corresponding positions as well as the union of all their attribute-value pairs.

Constructs

$$
\begin{array}{rcl}
m\_document & ::= & \mathsf{document}\ (m\_bundle \mid m\_term)^* \\
m\_bundle & ::= & \mathsf{bundle}(\Theta_i, m\_term^*) \\
m\_term & ::= & m\_term_1 \mid m\_term_2 \mid m\_term_3 \mid m\_term_4 \\
m\_term_1 & ::= & node(\Theta_i, [\kappa_i = \eta, \ldots, \kappa_j = \pi, \ldots]) \\
m\_term_2 & ::= & rel_1(\Theta_o, \Theta_e, \Theta_r, [\kappa_i = \eta, \ldots, \kappa_j = \pi, \ldots]) \\
m\_term_3 & ::= & rel_2(\Theta_o, \Theta_e, \Theta_r, \Theta_1, [\kappa_i = \eta, \ldots, \kappa_j = \pi, \ldots]) \\
m\_term_4 & ::= & rel_3(\Theta_o, \Theta_e, \Theta_r, \Theta_1, \Theta_2, \Theta_3, [\kappa_i = \eta, \ldots, \kappa_j = \pi, \ldots]) \\
m\_document & \in & M\_Document \\
m\_bundle & \in & M\_Bundle \\
m\_term & \in & M\_Term
\end{array}
$$

with $node, rel_1, rel_2, rel_3$ defined in Figure 1, and with $\Theta \subseteq \mathbb{N}$.

Positional roles

Θ_e	influencee
Θ_r	influencer
Θ_i	mandatory identifier
Θ_o	optional identifier
$\Theta_{1,2,3}$	optional argument

Accessors

	$id\downarrow$	$kind\downarrow$	$terms\downarrow$
m_bundle	Θ_i	bundle	m_term^*
m_term_1	Θ_i	$node$	
m_term_2	Θ_o	rel_1	
m_term_3	Θ_o	rel_2	
m_term_4	Θ_o	rel_3	

Fig. 2. Mergeable PROV

Likewise, PCO24 (unique-generation) stipulates that the pair of parameters η_e, η_r acts a key for the binary relation $rel_1 = \mathsf{wasGeneratedBy}$, which implies that the other parameters should be mergeable in a pair-wise fashion; in particular, the optional identifiers should be mergeable.

From this, we can define a $merge$ operation, such that Mergeable PROV is closed under the merge operation, meaning that the merge of two mergeable constructs is also a mergeable construct. (This contributes towards Requirement 2.4.) The conditions under which the merge operation is triggered are described in Section 4.4.

Definition 4.1 (*merge*). The binary operator $merge$ takes two mergeable constructs and returns a mergeable construct. Two constructs with the same predicate can be merged by taking the union of parameters in corresponding positions; we provide the definition for nodes (1) and terms of category $term_2$ (2). Bundles (3) can be merged if they have the same set of identifiers.

$$
\begin{array}{l}
merge(node(\Theta_i^1, attr^1), node(\Theta_i^2, attr^2)) \\
\quad = node(\Theta_i^1 \cup \Theta_i^2, attr^1 \cup attr^2)
\end{array}
\tag{1}
$$

$$\begin{aligned}&merge(rel_1(\Theta_o^1,\Theta_e^1,\Theta_r^1,attr^1),rel_1(\Theta_o^2,\Theta_e^2,\Theta_r^2,attr^2)) \\ &\ = \ rel_1(\Theta_o^1\cup\Theta_o^2,\Theta_e^1\cup\Theta_e^2,\Theta_r^1\cup\Theta_r^2,attr^1\cup attr^2)\end{aligned} \qquad (2)$$

$$\begin{aligned}&merge(\mathsf{bundle}(\Theta_i,terms^1),\mathsf{bundle}(\Theta_i,terms^2)) \\ &\ = \ \mathsf{bundle}(\Theta_i,terms^1\cup terms^2)\end{aligned} \qquad (3)$$

4.3. PROV Inferences

The PROV-CONSTRAINTS specification defines inferences over provenance documents. To ensure invariance to PROV inferences (Requirement 2.2), PROV-CONSTRAINTS inferences have to be re-defined over mergeable terms. We outline them here.

Type Inference. Section 2.1 already discusses type inference PCO50 (typing). We sketch its application to mergeable terms: whenever there is a term $\mathsf{wasGeneratedBy}(\Theta_o,\Theta_e,\Theta_r,attr)$, we can infer $\mathsf{entity}(\Theta_e,[\,])$ and $\mathsf{activity}(\Theta_r,[\,])$, meaning that there is an entity denoted by identifiers in Θ_e, and an activity, denoted by identifiers in Θ_r. Types are to be inferred similarly to all applicable cases.

WasInformedBy Inference. According to PCO6 (generation-use-communication-inference), for $\mathsf{wasGeneratedBy}(\Theta_o^1,\Theta_e^1,\Theta_a^1,attr^1)$ and $\mathsf{used}(\Theta_o^2,\Theta_a^2,\Theta_e^1,attr^2)$, one can infer $\mathsf{wasInformedBy}(\emptyset,\Theta_a^2,\Theta_a^1,[\,])$, meaning that the activity denoted by Θ_a^2 was informed by the activity Θ_a^1.

SpecializationOf, AlternateOf. Relation SpecializationOf is irreflexive and transitive. Relation AlternateOf is reflexive, symmetric, and transitive; furthermore, AlternateOf can be inferred from revisions (a subtype of derivation) and SpecializationOf. The adaptation of the following inferences to mergeable terms is straightforward: PCO12 (revision-is-alternate-inference), PCO16 (alternate-reflexive), PCO17 (alternate-transitive), PCO18 (alternate-symmetric), PCO19 (specialization-transitive), PCO20 (specialization-alternate-inference).

WasInfluencedBy. According to PCO15 (influence-inference), any relation is also an influence: Thus, for instance, $\mathsf{wasGeneratedBy}(\Theta_0,\Theta_a,\Theta_e,attr)$ also implies $\mathsf{wasInfluencedBy}(\Theta_0,\Theta_a,\Theta_e,attr)$.

All the above inferences are being applied to mergeable terms (M_Term) to meet Requirement 2.2. However, Inference PCO5 (communication-generation-use-inference), referred to as a completion rule [Kwasnikowska et al. 2015], infers the existence of a communicated entity. But neither PROV, nor the PROV Mergeable Syntax offers an existential quantifier or blank nodes to express this inference, so it is not applicable here. To sum up, we introduce a function $inference$ to capture all the above inferences over mergeable documents, with the signature: $inference : M\_Document \rightarrow M\_Document$. Full details about inferences are available in supplementary online material.

4.4. PROV Canonical Form

PROV Canonical Form is a mergeable PROV construct (or set of constructs) that is invariant to common PROV inferences (Section 4.3) and PROV-CONSTRAINTS key properties (Section 2). We provide a function that can convert a PROV document into its canonical form. It consists of two key steps, the mapping of a PROV document to PROV mergeable form, followed by its reduction to canonical form.

4.4.1. Injection. The mapping of a PROV document to the PROV mergeable form is straightforward: it just requires each name occurring as a parameter of a construct to be replaced by a set containing that name. We note that some names $\eta_o,\eta_1,\eta_2,\eta_3$ are optional. If unspecified, they are mapped to the empty set; if specified, they are mapped to singleton sets. (This behavior is expressed by $toSet$.)

$$\begin{array}{rcl}
inject & : & Document \rightarrow M\_Document \\
inject(\mathsf{document}\ (t^*)) & = & \mathsf{document}\ map(t^*, inject)) \\
inject(node(\eta_i, attr)) & = & node(\{\eta_i\}, attr) \\
inject(rel_1(\eta_o, \eta_e, \eta_r, attr)) & = & rel_1(toSet(\eta_o), \{\eta_e\}, \{\eta_r\}, attr) \\
inject(rel_2(\eta_o, \eta_e, \eta_r, \eta_1, attr)) & = & rel_2(toSet(\eta_o), \{\eta_e\}, \{\eta_r\}, toSet(\eta_1), attr) \\
inject(rel_3(\eta_o, \eta_e, \eta_r, \eta_1, \eta_2, \eta_3, attr)) & = & rel_3(toSet(\eta_o), \{\eta_e\}, \{\eta_r\}, toSet(\eta_1), toSet(\eta_2), toSet(\eta_3), attr) \\
inject(\mathsf{bundle}(\eta_i, terms^*)) & = & \mathsf{bundle}(\{\eta_i\}, map(terms^*, inject))
\end{array}$$

where $toSet(-) = \emptyset$ and $toSet(\eta) = \{\eta\}$.

4.4.2. Rearrange. According to Constraint PCO23 (key-properties), a key for a relation uniquely determines the other parameters of that relation. Thus, the two terms $\mathsf{wasGeneratedBy}(\{gen_0\}; \{e_1\}, \{a\}, \emptyset)$ and $\mathsf{wasGeneratedBy}(\{gen_0\}; \{e_2\}, \{a\}, \emptyset)$, have the same key $\{gen_0\}$, and can be merged into $\mathsf{wasGeneratedBy}(\{gen_0\}; \{e_1, e_2\}, \{a\}, \emptyset)$, which signifies that e_1 and e_2 are regarded as equivalent. To propagate the equivalence of e_1 and e_2 to the whole document, we assume that we have computed the equivalence relation between names, which partitions the set of names into subsets of equivalent names. From this relation, we define a *membership function* Ψ, which is a partial function associating a name with the set of terms it is equivalent with. Thus, we define a general rearrange function $\mathcal{R}_\mathcal{S}$ that replaces each name of a document with a set of names, relying on a function $\mathcal{S}$ mapping names to sets of names.

$$\begin{array}{rcl}
\mathcal{S} & : & \mathcal{N} \rightharpoonup \mathbb{S}et(\mathcal{N}) \\
\mathcal{R}_\mathcal{S} & : & M\_Document \rightarrow M\_Document \\
\mathcal{R}_\mathcal{S}(\mathsf{document}\ (t^*)) & = & \mathsf{document}\ (map(t^*, \mathcal{R}_\mathcal{S})) \\
\mathcal{R}_\mathcal{S}(node(\Theta, attr)) & = & node(fmap(\Theta, \mathcal{S}), fmap(attr, \mathcal{R}_\mathcal{S})) \\
\mathcal{R}_\mathcal{S}(rel_1(\Theta_o, \Theta_e, \Theta_r, attr)) & = & rel_1(fmap(\Theta_o, \mathcal{S}), fmap(\Theta_e, \mathcal{S}), fmap(\Theta_r, \mathcal{S}), fmap(attr, \mathcal{R}_\mathcal{S})) \\
\mathcal{R}_\mathcal{S}(rel_2(\Theta_o, \Theta_e, \Theta_r, \Theta_1, attr)) & = & rel_2(fmap(\Theta_o, \mathcal{S}), fmap(\Theta_e, \mathcal{S}), fmap(\Theta_r, \mathcal{S}), fmap(\Theta_1, \mathcal{S}), fmap(attr, \mathcal{R}_\mathcal{S})) \\
\mathcal{R}_\mathcal{S}(rel_3(\Theta_o, \Theta_e, \Theta_r, \Theta_1, \Theta_2, \Theta_3, attr)) & = & rel_3(fmap(\Theta_o, \mathcal{S}), fmap(\Theta_e, \mathcal{S}), fmap(\Theta_r, \mathcal{S}), fmap(\Theta_1, \mathcal{S}), fmap(\Theta_2, \mathcal{S}), \\
 & & \quad fmap(\Theta_3, \mathcal{S}), fmap(attr, \mathcal{R}_\mathcal{S})) \\
\mathcal{R}_\mathcal{S}(\mathsf{bundle}(\Theta_i, terms^*)) & = & \mathsf{bundle}(fmap(\Theta_i, \mathcal{S}), map(terms^*, \mathcal{R}_\mathcal{S})) \\
\mathcal{R}_\mathcal{S}(\kappa = \pi) & = & \{\kappa = \pi\} \\
\mathcal{R}_\mathcal{S}(\kappa = \eta) & = & map(\mathcal{S}(\eta), \lambda\eta'.\kappa = \eta')
\end{array}$$

4.4.3. Finding Equivalent Names. The presence of a set of names Θ in a construct indicates that the names in that set are regarded as equivalent. The function $\mathcal{F}$ computes the set of all sets of equivalent names in a mergeable m_term or m_bundle.

$$\begin{array}{rcl}
\mathcal{F} & : & M\_Term \cup M\_Bundle \rightarrow \mathbb{S}et(\mathbb{S}et(\mathcal{N})) \\
\mathcal{F}(node(\Theta, attr)) & = & \{\Theta\} \\
\mathcal{F}(rel_1(\Theta_o, \Theta_e, \Theta_r, attr)) & = & \{\Theta_o, \Theta_e, \Theta_r\} \\
\mathcal{F}(rel_2(\Theta_o, \Theta_e, \Theta_r, \Theta_1, attr)) & = & \{\Theta_o, \Theta_e, \Theta_r, \Theta_1\} \\
\mathcal{F}(rel_3(\Theta_o, \Theta_e, \Theta_r, \Theta_1, \Theta_2, \Theta_3, attr)) & = & \{\Theta_o, \Theta_e, \Theta_r, \Theta_1, \Theta_2, \Theta_3\} \\
\mathcal{F}(\mathsf{bundle}(\Theta_i, terms^*)) & = & \{\Theta_i\} \cup fmap(terms^*, \mathcal{F})
\end{array}$$

To merge terms, we define a general function $index\_membership(t^*, key, active, nn)$ that indexes a set of terms $t^* : \mathbb{S}et(M\_Term)$ according to their type $Kind$ and some key

$key : (M\_Term \rightarrow Object)$, resulting in an index $indx : ((Kind \times Object) \rightarrow \mathbb{S}et(M\_Term))$ and a membership function $\Psi : (\mathcal{N} \rightharpoonup \mathbb{S}et(\mathcal{N}))$.

To construct an index, we use the function $groupBy$ that partitions a set into a map of sets according to some discriminator function $\lambda t.\langle kind \downarrow (t), key(t)\rangle$, which pairs up the kind of a term with its parametric key. The set of terms are then merged into a single term, by application of $reduce$ to $merge$. Since not all the terms necessarily have a key to index on, a predicate $active$, passed as argument to $index\_membership$, is used to decide whether to apply $reduce$ to the appropriate terms.

$$\begin{array}{rcl}
index\_membership & : & \mathbb{S}et(M\_Term) \times (M\_Term \rightarrow Object) \\
& & \times\ (Kind \times Object \rightarrow Boolean)\ \times\ \mathbb{S}et(\mathcal{N}) \\
& & \rightarrow ((Kind \times Object) \rightarrow \mathbb{S}et(M\_Term)) \times (\mathcal{N} \rightharpoonup \mathbb{S}et(\mathcal{N}))) \\
index\_membership(t^*, key, active, nn) & = & \textbf{let}\ indx = map(\ groupBy(t^*, \lambda t.\langle kind \downarrow (t), key(t)\rangle), \\
& & \qquad \lambda\langle p, l\rangle.\langle p, \textbf{if}\ active(p)\ \textbf{then}\{reduce(l, merge)\}\ \textbf{else}\ l\rangle) \\
& & \textbf{let}\ \Psi = make\_membership(fmap(indx, \lambda\langle k, t\rangle.\mathcal{F}(t))\ \cup\ nn) \\
& & \textbf{in}\ \langle indx, \Psi\rangle \\
index\_membership_{i,k} & : & \mathbb{S}et(M\_Term) \rightarrow (\mathcal{N} \rightarrow \mathbb{S}et(\mathcal{N})) \\
index\_membership_i(t^*) & = & index\_membership(t^*, id\downarrow, \lambda\langle kind, id\rangle.id \neq \emptyset, \emptyset) \\
index\_membership_k(t^*) & = & index\_membership(t^*, key, \lambda\langle kind, key\rangle.key \neq false, \emptyset)
\end{array}$$

The function $index\_membership$ is instantiated in two different ways: $index\_membership_i$ is indexing terms by their identifiers (implementing PCO23 (key-properties)), whereas $index\_membership_k$ indexes those relations that have a compound key (implementing PCO24 (unique-generation), PCO25 (unique-invalidation), PCO26 (unique-start), and PCO27 (unique-end)). For the latter, the auxiliary function key defines a *compound key* for terms wasGeneratedBy, wasInvalidatedBy, wasStartedBy, wasEndedBy, as follows.

$$\begin{array}{rcl}
key(rel_1(\Theta_o, \Theta_e, \Theta_r, attr)) & = & \langle\Theta_e, \Theta_r\rangle\ \text{if}\ rel_1 \in \{\text{wasGeneratedBy}, \text{wasInvalidatedBy}\},\ \Theta_e \neq \emptyset,\ \text{and}\ \Theta_r \neq \emptyset \\
key(rel_2(\Theta_o, \Theta_e, \Theta_r, \Theta_1, attr)) & = & \langle\Theta_e, \Theta_1\rangle\ \text{if}\ rel_2 \in \{\text{wasStartedBy}, \text{wasEndedBy}\},\ \Theta_e \neq \emptyset,\ \text{and}\ \Theta_r \neq \emptyset \\
key(\_) & = & false\ \ \text{otherwise}
\end{array}$$

The function $index\_membership$ relies on $make\_membership$ to compute the transitive closure of the equivalence relation and creates a membership function Ψ.

$$\begin{array}{rcl}
make\_membership & : & \mathbb{S}et(\mathbb{S}et(\mathcal{N})) \rightarrow (\mathcal{N} \rightharpoonup \mathbb{S}et) \\
make\_membership(ss^*) & = & fmap(transitive\_closure(ss^*), \lambda ss.map(ss, \lambda n.\langle s, ss\rangle))
\end{array}$$

4.4.4. Term Fusion. PROV-CONSTRAINTS defines an algorithm based on unification to determine a normal form NF in the process of validating expressions; an implementation of this approach [Moreau et al. 2014] showed that full unification was not required to compute NF. Here, instead of relying on unification, we define a procedure $fusion$ that applies a membership function Ψ using the rearrange function of Section 4.4.2. In a first instance, we create an index with term identifiers.

$$\begin{array}{rcl}
fusion_i & : & \mathbb{S}et(M\_Term) \rightarrow \mathbb{S}et(M\_Term) \\
fusion_i(t^*) & = & \textbf{let}\ \langle indx, \Psi\rangle = index\_membership_i(t^*) \\
& & \textbf{in}\ fmap(indx.values, \lambda t.\mathcal{R}_\Psi(t))
\end{array}$$

In a second instance, we index terms that have a compound key (obtained by key):

$$\begin{array}{rcl} fusion_k & : & \mathbb{S}et(M\_Term) \to \mathbb{S}et(M\_Term) \\ fusion_k(t^*) & = & \textbf{let } \langle indx, \Psi\rangle = index\_membership_k(t^*) \\ & & \textbf{in } fmap(indx.values, \lambda t.\mathcal{R}_\Psi(t)) \end{array}$$

Such a succession of operations is repeated until we converge to a canonical form, for which we introduce the type PCF_Term, specified in Section 4.4.5.

$$\begin{array}{rcl} fusion & : & \mathbb{S}et(M\_Term) \to \mathbb{S}et(PCF\_Term) \\ fusion(t^*) & = & \textbf{let } t_1^* = fusion_k(fusion_i(t^*)) \\ & & \textbf{in if } t_1^* = t^* \textbf{ then } t^* \textbf{ else } fusion(t_1^*) \end{array}$$

4.4.5. Transformation To Canonical Form. We can now specify the transformation of a PROV document to a document in canonical form ($PCF\_Document$). First, we define the notion of canonical form, which is invariant to PROV inferences as per Requirement 2.2.

Definition 4.2 (*Canonical Form*). A set of terms t^* (a document d) is in canonical form, noted $t^* \in \mathbb{S}et(PCF\_Term)$ ($d \in PCF\_Document$), if it is in mergeable form $t^* \in \mathbb{S}et(M\_Term)$ ($d \in M\_Document$) and is invariant to inference ($inference$) and fusion ($fusion$).

The transformation to canonical form requires a document to be mapped to its mergeable form, all inferences performed, and fusion applied.

$$\begin{array}{rcl} canonical & : & Document \to PCF\_Document \\ canonical(d) & = & fusion(inference(inject(d))) \end{array}$$

We can establish the unique existence of a canonical form for a set of terms.

THEOREM 4.3 (CANONICAL FORM EXISTENCE). *For any set of terms, there exists a unique canonical form.*

PROOF. The proof can be sketched as follows. Each iterative step of $fusion$ involves a partition of the set of names. At the start, before any iteration has taken place, each name is only equivalent to itself. Thus, the number of subsets in the partition is given by the number of names. Each fusion step strictly reduces the number of terms, the number of name subsets, or both. There is a lower bound, which minimally consists of a single term and a single subset of names, by which all names are regarded as equivalent. Thus, there can only be a finite number of iterations until a fixpoint is reached. Furthermore, $canonical$ is deterministic and therefore results in a unique fixpoint. Such a fixpoint satisfies the definition of canonical form, and thus Requirements 2.1, 2.2, and 2.4. □

Supplementary material explains how bundles are supported by the $fusion$ function.

To illustrate PROV Canonical Form, Figure 3 shows a PROV document. The canonization process merges the two instances of entities, the generations with identifier `ex:gen10` (with $fusion_i$), and the generations with identifiers `ex:gen10` and `ex:gen20` (with $fusion_k$). The PROV canonical form, also displayed in Figure 3, consists of a set of three terms (for clarity, we have not included inferred `wasInfluencedBy` edges).

4.5. Canonical Form Equality

Equality on mergeable terms, and therefore on canonical forms, forms the essence of Requirement 2.3. For two terms to be equal, their corresponding constituents must be equal and they must have the same kind. For instance, for terms of type m_term_2, equality is defined as follows.

$$\begin{array}{l} rel_1^1(\Theta_o^1, \Theta_e^1, \Theta_r^1, attr^1) = rel_1^2(\Theta_o^2, \Theta_e^2, \Theta_r^2, attr^2) \\ \text{ if } \Theta_o^1 = \Theta_o^2,\ \Theta_e^1 = \Theta_e^2,\ \Theta_r^1 = \Theta_r^2,\ attr^1 = attr^2, \text{ and } rel_1^1 = rel_1^2 \end{array}$$

Section 5.1 explains how such a notion of equality can be used to establish the equality of two documents serialized in different formats.

4.6. Canonical Form Serialization

For the purpose of signing documents, we need to be able to produce a serialization of their canonical form that can be signed. An option is to convert PROV canonical documents to a representation that already has a canonical form, contruct such a canonical form and sign it; one such representation is RDF, which is already equipped with a canonical form [Longley and Sporny 2016; Carroll 2003]. Alternatively, we can specify a unique serialization, based on some ordering of mergeable terms, to which we can apply a signature algorithm. We opted for the latter option for a pragmatic reason, given the existence of a normative specification [Eastlake et al. 2008] and availability of libraries to sign XML documents. For this, first, we define a lexicographic order on mergeable terms. The choice of order is arbitrary: its only requirement is to enable unique serialization. Lexicographic order is adopted because it is easy to implement.

$$\begin{array}{l} \text{entity} \prec \text{activity} \prec \text{agent} \prec \text{wasDerivedFrom} \prec \text{wasGeneratedBy} \prec \text{used} \prec \text{wasAttributedTo} \\ \quad \prec \text{wasInvalidatedBy} \prec \text{wasInformedBy} \prec \text{wasInfluencedBy} \prec \text{wasStartedBy} \prec \text{wasEndedBy} \\ \quad \prec \text{wasAssociatedWith} \prec \text{actedOnBehalfOf} \prec \text{specializationOf} \prec \text{alternateOf} \prec \text{hadMember} \\ rel_i(\Theta_0^1, \ldots, \Theta_k^1, attr^1) \prec rel_j(\Theta_o^2, \ldots, \Theta_l^2, attr^2) \quad k \neq \ell \\ \ \text{ if } rel_i \prec rel_j \\ rel_i(\ldots, \Theta_k^1, \ldots, \Theta_m^1, attr^1) \prec rel_j(\ldots, \Theta_k^2, \ldots, \Theta_m^2, attr^2) \\ \ \text{ if } rel_i \prec rel_j \\ \ \text{ or } rel_i = rel_j,\ \Theta_\ell^1 = \Theta_\ell^2, \text{ for } \ell \in [0, k-1], \text{ and } \Theta_k^1 \prec \Theta_k^2 \\ \ \text{ or } rel_i = rel_j,\ \Theta_\ell^1 = \Theta_\ell^2, \text{ for } \ell \in [0, m], \text{ and } attr^1 \prec attr^2 \end{array}$$

Two sets Θ^1, Θ^2 are lexicographically ordered, $\Theta^1 \prec \Theta^2$, if the lexicographically ordered sequences of names from Θ^1 and Θ^2 are lexicographically ordered. Likewise, sets of attributes can be converted into lexicographically-ordered sequences of key-value pairs, and therefore can also be ordered lexicographically.

Thus, to construct a serialization of a canonical form, we just need to enumerate all terms in their lexicographic order, with each set of names or attributes, also serialized by enumerating their elements in lexicographic order. This addresses Requirement 2.5.

We revisit Figure 3, which for a PROV document, displays its PROV Canonical Form and the canonical form's XML serialization. All names have been expanded to full URIs (to avoid dependencies on prefix declarations), and terms, names, and attributes have been lexicographically ordered. An attribute `Id` was added to the `document` element, as a means to identify the element subject to signature (see Section 5.2).

5. APPLICATIONS OF PROV CANONICAL FORM

In this section, we discuss applications of PROV Canonical Form to serialization equality (Section 5.1), signature (Section 5.2) and validation (Section 5.3).

```
document
 prefix ex <http://example/>
 entity(ex:e10, [prov:value=1])
 entity(ex:e10, [ex:foo="a"])
 wasGeneratedBy(ex:gen10;
                ex:e10,ex:a1,-)
 wasGeneratedBy(ex:gen10;
                ex:e20,ex:a1,-)
 wasGeneratedBy(ex:e10,ex:a1,-,
                [ex:foo=1])
 wasGeneratedBy(ex:gen20;
                ex:e20,ex:a1,-)
 wasGeneratedBy(ex:e20,ex:a100,-)
endDocument
```

$$
\begin{array}{l}
\{\\
\text{entity}(\ \{e10, e20\},\\
\qquad \{\text{prov:value} = 1, \text{ex:foo} = \text{"a"}\})\\
\text{wasGeneratedBy}(\ \{gen10, gen20\},\\
\qquad\qquad \{e10, e20\},\\
\qquad\qquad \{a1\},\\
\qquad\qquad \{\text{ex:foo} = 1\})\\
\text{wasGeneratedBy}(\ \emptyset,\\
\qquad\qquad \{e10, e20\},\\
\qquad\qquad \{a100\},\\
\qquad\qquad \emptyset)\\
\}
\end{array}
$$

```
<?xml version="1.0" encoding="UTF-8"?>
<document Id="id12345">
 <entity>
  <id>http://example/e10</id>
  <id>http://example/e20</id>
  <attr>
    <element>http://example/foo</element>
    <value>a</value>
    <type>http://www.w3.org/2001/XMLSchema#string</type>
  </attr>
  <attr>
    <element>http://www.w3.org/ns/prov#value</element>
    <value>1</value>
    <type>http://www.w3.org/2001/XMLSchema#int</type>
  </attr>
 </entity>
 <wasGeneratedBy>
   <entity>http://example/e10</entity>
   <entity>http://example/e20</entity>
   <activity>http://example/a100</activity>
 </wasGeneratedBy>
 <wasGeneratedBy>
   <id>http://example/gen10</id>
   <id>http://example/gen20</id>
   <entity>http://example/e10</entity>
   <entity>http://example/e20</entity>
   <activity>http://example/a1</activity>
   <attr>
     <element>http://example/foo</element>
     <value>1</value>
     <type>http://www.w3.org/2001/XMLSchema#int</type>
   </attr>
 </wasGeneratedBy>
</document>
```

Fig. 3. A PROV document (top left), its PROV Canonical Form (bottom left) and a serialization of its canonical form in XML (right)

5.1. Application 1: Equality Across Serializations

To establish equality of documents in different serialization formats, we need to define a mapping of each serialization to PROV Canonical Form. (For conversion between formats, we further need mappings from PROV Canonical Form to each serialization format.) It is simple to map the "expression-oriented" serializations such as PROV-N [Moreau et al. 2013], PROV-XML [Hua et al. 2013], and PROV-JSON [Huynh et al. 2013] to mergeable terms M_Term, with a view of determining their canonical form.

Instead, we focus on the mapping of RDF to PROV Canonical Form. It is more challenging because atomic information in RDF is expressed by triples, and it may take several triples to encode a PROV term. ProvRDF [ProvRDF 2013] is an incomplete mapping of RDF to PROV-DM. We revisit this mapping, using mergeable terms, affording more flexibility than PROV terms. Figure 4 displays the possible mapping for various forms of generations. The mapping has been decomposed in different categories. (1) is concerned with the unqualified generation, expressed with a single property; we see that the mergeable term $\text{wasGeneratedBy}(\emptyset, \{e\}, \{a\}, \emptyset)$ refers to the entity e and activity a, respectively subject and object of property prov:wasGeneratedBy. (2) is concerned with the mapping involving a qualified generation with an unlabeled blank node [Beckett et al. 2014] for the generation. (3) is describing the mapping for a qualified generation with an explicit blank node $\_ : id$, which does not appear in the PROV canonical form since identifiers are expected to be well-formed URIs. (4) deals with a qualified generation with an explicit identifier. (5) and (6) have in common that no entity is part of the RDF description: (5) deals with generation with a blank node, whereas

(6) is concerned with the case of a generation with an explicit identifier. Finally, (7) and (8) deal with the degenerate cases in which no entity nor activity is mentioned.

	RDF Triples	M_Term
1	e wasGeneratedBy a	wasGeneratedBy$(\emptyset, \{e\}, \{a\}, \emptyset)$
	e qualifiedGeneration [activity a].	wasGeneratedBy$(\emptyset, \{e\}, \{a\}, \emptyset)$
2	e qualifiedGeneration [atTime t].	$(\dagger)$wasGeneratedBy$(\emptyset, \{e\}, \emptyset, \{$prov:time $= t\})$
	e qualifiedGeneration [activity a; atTime t].	wasGeneratedBy$(\emptyset, \{e\}, \{a\}, \{$prov:time $= t\})$
	e qualifiedGeneration _:id. _:id activity a.	wasGeneratedBy$(\emptyset, \{e\}, \{a\}, \emptyset)$
3	e qualifiedGeneration _:id. _:id atTime t.	$(\dagger)$wasGeneratedBy$(\emptyset, \{e\}, \emptyset, \{$prov:time $= t\})$
	e qualifiedGeneration _:id. _:id atTime t. _:id activity a.	wasGeneratedBy$(\emptyset, \{e\}, \{a\}, \{$prov:time $= t\})$
	e qualifiedGeneration id. id activity a.	wasGeneratedBy$(\{id\}, \{e\}, \{a\}, \emptyset)$
4	e qualifiedGeneration id. id atTime t	wasGeneratedBy$(\{id\}, \{e\}, \emptyset, \{$prov:time $= t\})$
	e qualifiedGeneration id. id atTime t. id activity a.	wasGeneratedBy$(\{id\}, \{e\}, \{a\}, \{$prov:time $= t\})$
	_:id a Generation; activity a.	$(\dagger)$wasGeneratedBy$(\emptyset, \emptyset, \{a\}, \emptyset)$
5	_:id a Generation; atTime t.	$(\dagger)$wasGeneratedBy$(\emptyset, \emptyset, \emptyset, \{$prov:time $= t\})$
	_:id a Generation; atTime t; activity a.	$(\dagger)$wasGeneratedBy$(\emptyset, \emptyset, \{a\}, \{$prov:time $= t\})$
	id a Generation; activity a.	wasGeneratedBy$(\{id\}, \emptyset, \{a\}, \emptyset)$
6	id a Generation; atTime t.	wasGeneratedBy$(\{id\}, \emptyset, \emptyset, \{$prov:time $= t\})$
	id a Generation; activity a; atTime t.	wasGeneratedBy$(\{id\}, \emptyset, \{a\}, \{$prov:time $= t\})$
	_:id a Generation; activity a.	$(\dagger)$wasGeneratedBy$(\emptyset, \emptyset, \emptyset, \emptyset)$
7	_:id a Generation; atTime t.	$(\dagger)$wasGeneratedBy$(\emptyset, \emptyset, \emptyset, \{$prov:time $= t\})$
	id a Generation.	wasGeneratedBy$(\{id\}, \emptyset, \emptyset, \emptyset)$
8	id a Generation; atTime t.	wasGeneratedBy$(\{id\}, \emptyset, \emptyset, \{$prov:time $= t\})$

Fig. 4. Mapping from RDF to Mergeable Terms. All properties and classes are in the PROV namespace.

While Figure 4 exhaustively considers cases for generations, we note that the terms marked with $(\dagger)$ cannot be merged with any other term by $fusion$, because they do not have an identifier and their influencer or influencee is empty, and so are not indexable with $index\_membership_{i,k}$. It is, however, useful to handle these cases because they allow us to define round-trip conversions and to reason about the conditions under which such round-trip conversions preserve their inputs.

Other PROV relations can be mapped in a similar manner. With the mapping to RDF now defined, we can map a PROV document to its canonical form, for all serializations. We can then define equality of two documents d_1, d_2 in respective serialization formats f_1, f_2, in terms of the equality of their respective canonical forms.

5.2. Application 2: Signature

So far, we have defined a canonical form for PROV documents and a serialization obtained by enumerating all terms of the canonical form in lexicographic order. Section 4.6 has further presented a unique serialization of the canonical form.

A unique serialization is a necessary prerequisite to a signature mechanism. The mechanism for signature needs to be made public (ideally, standardized) for third-parties to be able to perform it independently. An obvious approach, building on existing standards, is to apply XML Signature [Eastlake et al. 2008] to obtain a signature of the XML serialization of PROV Canonical Form. An example of signature is contained in Supplementary Material. The contents of such an XML Signature element consist of all the information required by a third party to check the signature of a document. It could be embedded inside a PROV document to form a *signed* PROV document.

Alternative serialization methods and signatures could be adopted, for instance, based on the nacent JSON normalization and signature [Sporny 2015; Longley and

Sporny 2016] as long as the representation of the signature contains all the necessary information for a third-party to determine how to verify a signature.

5.3. Application 3: Validation

The PROV-CONSTRAINTS specification [Cheney et al. 2013] defines a set of PROV terms as *valid* "if their normal form exists and all of the validity constraints succeed on the normal form": this ensures that these terms present a logically consistent history, meaning that a logical interpretation of these terms can be found. By converting terms to PROV Canonical Form, we have already implemented part of the validation process. We now examine the steps necessary to carry out validation of terms in PROV Canonical Form.

The following constraints cater for PCO22 (key-object), PCO23 (key-properties), PCO24 (unique-generation), PCO25 (unique-invalidation), PCO26 (unique-start), and PCO27 (unique-end).

CONSTRAINT 5.1 (SET SIZE). *Let* $t^* \in \mathbb{S}et(PCF\_Term)$*, for any* $\Theta, \Theta_0, \Theta_r, \Theta_e, \Theta_{1,2,3}, attr$ *such that:*

— *if* $node(\Theta, attr) \in t^*$*, then* $|\Theta| = 1$
— *if* $rel_1(\Theta_0, \Theta_e, \Theta_r, attr) \in t^*$*, then* $|\Theta_0| \leq 1$ *and* $|\Theta_e| = |\Theta_r| = 1$
— *if* $rel_2(\Theta_0, \Theta_e, \Theta_r, \Theta_1, attr) \in t^*$*, then* $|\Theta_0| \leq 1$ *and* $|\Theta_e| = |\Theta_r| = 1$ *and* $|\Theta_1| \leq 1$.
— *if* $rel_3(\Theta_0, \Theta_e, \Theta_r, \Theta_1, \Theta_2, \Theta_3, attr) \in t^*$*, then* $|\Theta_0| \leq 1$ *and* $|\Theta_e| = |\Theta_r| = 1$ *and* $|\Theta_i| \leq 1$.

We see that the example of Figure 3 does not satisfy Constraint 5.1 because the entity has two identifiers e10 and e20.

Some attributes, such as those related to time, or value, are expected to have one value at most (cf. PCO28 (unique-startTime), PCO29 (unique-endTime)).

CONSTRAINT 5.2 (UNIQUE ATTRIBUTES). *Let* $t^* \in \mathbb{S}et(PCF\_Term)$*, for any* $\Theta, \Theta_0, \Theta_r, \Theta_e, \Theta_{1,2,3}, attr$ *such that:*

— *if* $\mathsf{entity}(\Theta, attr) \in t^*$*, then* $|attr(\mathsf{prov{:}value})| \leq 1$.
— *if* $\mathsf{activity}(\Theta, attr) \in t^*$*, then* $|attr(\mathsf{prov{:}startTime})| \leq 1$, $|attr(\mathsf{prov{:}endTime})| \leq 1$.
— *if* $rel_1(\Theta_0, \Theta_e, \Theta_r, attr) \in t^*$*, then* $|attr(\mathsf{prov{:}time})| \leq 1$ *with* $rel_1 \in \{\mathsf{wasGeneratedBy}, \mathsf{wasInvalidatedBy}\}$.
— *if* $rel_2(\Theta_0, \Theta_e, \Theta_r, \Theta_1, attr) \in t^*$*, then* $|attr(\mathsf{prov{:}time})| \leq 1$ *with* $rel_2 \in \{\mathsf{wasStartedBy}, \mathsf{wasEndedBy}\}$.

PROV-CONSTRAINTS defines several impossibility constraints, such as PCO51 (impossible-unspecified-derivation-generation-use), PCO52 (impossible-specialization-reflexive), PCO53 (impossible-property-overlap), PCO54 (impossible-object-property-overlap), which are directly applicable to PROV Canonical Form. On the other hand, PCO55 (entity-activity-disjoint), which requires the set of entities and activities to be disjoint, can be reformulated as follows:

CONSTRAINT 5.3 (ENTITY ACTIVITY DISJOINT). *Let* $t^* \in \mathbb{S}et(PCF\_Term)$*, for any* $\Theta, \Theta', attr, attr'$ *such that* $\mathsf{entity}(\Theta, attr) \in t^*$ *and* $\mathsf{activity}(\Theta', attr') \in t^*$*, then* $\Theta \cap \Theta' = \emptyset$.

Type inference has already been performed as part of the *inference* transformation. Constraint PCO50 (typing) further deals with collections, which still need to be handled similarly. Likewise, PCO56 (membership-empty-collection) remains applicable.

Finally, PROV-CONSTRAINTS defines a series of ordering constraints: they should be applied similarly to PROV Canonical Form. A summary can be found in Supplementary Material, enumerating all contraints of PROV-CONSTRAINTS and how they are supported in our revised validation algorithm.

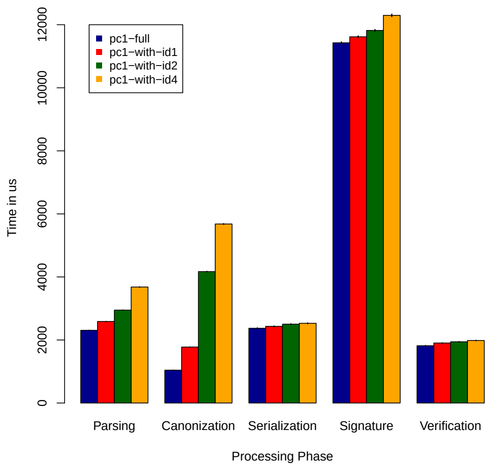
6. EVALUATION

The definition of $canonical$ presented in Section 4 was used to implement PROV Canonical Form in Scala. The aim of this section is to understand the relative costs of the various components involved in processing provenance. Specifically, we compare the time required for the following operations:

— Parsing: loading a file and parsing PROV-N representations;
— Canonization: the canonization procedure;
— Serialization: the procedure to serialize PROV Canonical Form into an XML file;
— Signature: the procedure to sign the XML serialization on the fly;
— Verification: the procedure to check if a signature is valid.

The performance evaluation was conducted on a MacBookPro 10.1, with an Intel Core i7, 2.7 GHz, and 16Gb of Memory, using the JMH benchmarking framework. Figure 5 displays the result of this evaluation.

We made use of 4 different files, whose characteristics are displayed in Figure 5 (right). `pc1-full` is a PROV-N encoding of the provenance of the Provenance Challenge FMRI workflow [Moreau and Ludaescher 2008]. `pc1-with-id1` is a variant in which all relations are given an identifier (column i shows the number of relations with identifiers); this dataset exercises $index\_membership_i$. `pc1-with-id2` duplicates generations, providing them with new identifiers (see column k), with a view of exercising $index\_membership_k$. Finally, `pc1-with-id4` quadruples the number of generations.



Relative Time to Parsing `pc1-full` (as %)

File	P	C	Se	Si	Ve
`pc1-full`	100	45	103	495	79
`pc1-with-id1`	112	77	105	504	83
`pc1-with-id2`	128	181	108	512	84
`pc1-with-id4`	160	246	110	533	86

File Characteristics

File	n	i	k
`pc1-full`	163	0	0
`pc1-with-id1`	163	110	20
`pc1-with-id2`	191	130	40
`pc1-with-id4`	232	170	80

n number of terms
i number of relations with an identifier
k number of generations with compound key

Fig. 5. Parsing (P), Canonization (C), Serialization (Se), Signing (Si), and Verification (Ve). Left: Absolute Time. Right: Time Relative to Parsing File `pc1-full`

With `pc1-full`, we see that reading is approximately 2.3ms, canonization 45% of parsing at 1ms; serialization 103%, signature about 500%, and verification about 80%.Canonization has a lower cost than parsing, itself about the same as serialization. Confidence intervals are displayed in Figure 5 (left), but are too small to be visible; for instance, the 99.9% interval for canonization is $1ms \pm 1.2\mu s$. Signing is performed on the fly, on the serialized stream: so the time to sign includes the time to serialize (potentially, executed in a separate thread). Checking the signature is a constant time operation. As the indexation effort and opportunities to fusion terms increase, the normalization time also increases. It is our expectation that normalization time increases faster than that of parsing: profiling our code indicates time is split three ways between indexing terms (complexity $O(dn\log(n))$, where $O(n\log(n))$ is for indexing and $O(d)$ for application of $reduce$), computing transitive closures of specialization (complexity $(O(spec^3))$) and applying membership function (complexity $(O(|\mathcal{N}|^3))$, with $n, spec, \mathcal{N}, d$ the number of specializations, variables, names, and mergeable terms, respectively. While the theoretical worst case complexity exhibits high-degree polynomials, we have observed that such cases have to be engineered (as in our evaluation) rather than occur in usually generated provenance.

The take-home message is that for documents that are essentially already in canonical form, the canonisation cost is a fraction of that of parsing. As the number of terms to be merged increases, the canonisation cost overtakes that of parsing, while still remaining a small fraction of signing a document.

7. RELATED WORK

This work focuses on PROV, a model of provenance that can be represented as graphs. For a survey of provenance in databases, we refer the reader to [Cheney et al. 2009]. In computer science, canonization (also canonicalization) is a process for converting data that has more than one possible representation into a common form. It is important to contrast PROV Canonical Form with “graph canonical forms” in graph theory.

In graph theory literature, graphs consist of unlabelled nodes and unlabelled edges. In graph theory, an isomorphism between two graphs G and H is a bijection f between the vertex sets of G and H such that any two vertices u and v of G are adjacent in G if and only if $f(u)$ and $f(v)$ are adjacent in H. The graph isomorphism problem consists of deciding whether two graphs are isomorphic. A common way of solving this problem is to determine if the canonical representations of those graphs are identical. A canonical representation is obtained by a canonization process: a canonical form of a graph G is a labeled graph that is isomorphic to G, such that every graph that is isomorphic to G has the same canonical form as G. McKay's “nauty” algorithm [McKay and Piperno 2014] is an example of graph canonization. The graph isomorphism problem is not known to be solvable in polynomial time, but is not known to be NP-complete either. The problem we tackle with PROV documents is different, since nodes are labelled (with their identifiers) and edges are also labelled (with their kinds and optional identifiers) and directed. For this reason, we opted to talk about PROV *documents*, and kept the term “graph” to refer to the eponymous notion in graph theory.

The ability to compare two RDF graphs for equality is a fundamental operation of the Semantic Web [Carroll 2002]. RDF graph equality is a specific instance of the graph isomorphism problem. Like PROV graphs, RDF graphs are also directed graphs, with labeled edges but partially labelled nodes (since blank nodes are allowed in RDF). Carroll's approach to an RDF graph canonical form is to apply the iterative vertex classification algorithm [Read and Corneil 1977].

To sign RDF graphs, however, Carroll [Carroll 2003] introduces Canonical RDF, consisting of lexicographically ordered triples, with an extra mechanism to “normalize” the names of blank nodes. The complexity of this operation is known to be $O(n\log(n))$. A more recent canonization algorithm is being developed by Longley and Sporny [Longley and Sporny 2016] for RDF datasets, with a view of defining a message signature algorithm as part of a secure messaging layer [Sporny 2015]. The signature itself is represented as a JSON dictionary, including the signature algorithm, the creator, the date of creation, and the signature value.

XML Signature [Eastlake et al. 2008] provides integrity and message authentication for XML documents. The processing rule and syntax of XML Signature meet some requirements [Reagle 2008]: XML-signature data structures are based on the

RDF data model. Multiple XML-signatures must be able to co-exist over the content of an XML Document given varied keys, content transformations, and algorithm specifications (signature, hash, canonicalization, etc.). XML-signatures are XML elements, and thus first class objects and consequently can be referenced and signed. While XML-Signature permits arbitrary cryptographic signature algorithms, it mandates support for signature canonicalization, content canonicalization, hash, and signature algorithm.

The canonical form of an XML document [Boyer and Marcy 2008] is a physical representation of the document that underwent several transformative steps. We enumerate a few that are relevant to this paper: the document is encoded in UTF-8; line breaks are normalized; namespace declarations and attributes of each element are lexicographically ordered; whitespace outside of the document element and within start and end tags is normalized; attribute value delimiters are set to double quotes.

Experience with writing converters (cf. ProvToolbox[4]) shows that each representation has its own quirks, and round-trip conversion between representations is not straightforward. Therefore, formalising conversions between representations is critical, and PROV Canonical Form is a significant step in this direction. In the context of standardisation, formalisations of PROV can be found in PROV-CONSTRAINTS [Cheney et al. 2013] and PROV-SEM [Cheney 2013]. The former specifies a notion of valid provenance using a set of logical declarative rules, whereas the latter consists of a model-theoretic semantics for PROV. The temporal constraints specified by PROV-CONSTRAINTS were inspired by an earlier formalisation [Kwasnikowska et al. 2015] of the Open Provenance Model (OPM) [Moreau et al. 2011].

While our work focuses on creating a canonical form suitable for signature, others have investigated the contents of the provenance to be signed so as to offer some guarantees to applications. Gadelha and Matthoso propose the Kairos architecture [Gadelha and Mattoso 2008] featuring a timestamping service, which combined with signature capability, offers a non-repudiable record of authorship and time for grid applications. Concerned by undetected rewrites of history, Hasan et al. [2009] propose a linear provenance record chaining technique, which embeds in a record the signature of its predecessors. This scheme is susceptible to attack on the most recent record, since it could be replaced by a new one pointing to the same predecessors. To avoid this type of attacks, the public-key linked chain [Wang et al. 2012] adds to a record the public key of the successor in the chain. Such techniques are important, for instance, in applications where audits [Aldeco-Pérez and Moreau 2010] have to ensure that the history has not been rewritten.

8. CONCLUDING REMARKS AND FUTURE WORK

Provenance documents can vary because of their syntactic representations or because of simple PROV inferences. We have presented PROV Canonical Form, which allows documents to be reduced to a common normalized representation, to be compared and to be signed. The canonization algorithm, converting a provenance document to its canonical form, is defined as a fixed point, demonstrated to exist for any PROV document. An empirical evaluation shows that the algorithm is tractable and its execution is only a fraction of input and output operations.

PROV Canonical Form has a number of important applications. First, it addresses a shortcoming of the standardization process of PROV, which defined a normal form within the context a validation procedure, but only for some provenance documents. So, with this work, validation can instead be defined on the canonical form. Furthermore, safe round-trip conversions can be defined using PROV Canonical Form.

Finally, one of the key drivers for this work is the need to sign provenance: provenance is only useful if it can be proven not to have been tampered with. We have provided the means to uniquely serialize PROV Canonical Form and apply standard cryptographic signature techniques to it.

This work presents further opportunities related to systems deployment and standardisation activities. Practically, a provenance signature service could be deployed: such a service could sign provenance documents and check the validity of signatures. It could further be integrated with a provenance repository such as ProvStore [Huynh and Moreau 2014]. It may be worthwhile to explore how a repository for signed provenance documents could further be combined with distributed ledger technology, allowing distributed proof-of-work to bring trust to a provenance repository.

However, for such an environment to succeed, inter-operability is required, so that the signing of documents and the checking of signatures can be performed by independent agents. A standardization group needs to specify the serialization of PROV Canonical Form, the transformations allowed to be applied prior to signature, the algorithms supported for signing, and the representation of the signature, and where it is allowed to be inserted in PROV documents.

Another important aspect of inter-operability is the conversion of all serializations of PROV to PROV canonical form, and vice-versa. To address it fully, one needs to further specify the necessary conventions to ensure that round-trip conversions are contents preserving.

A. DATA AVAILABILITY

The data supporting the evaluation of Section 6 are openly available from the University of Southampton repository at DOI: 10.5258/SOTON/402535.

REFERENCES

Rocio Aldeco-Pérez and Luc Moreau. 2010. Securing Provenance-based Audits. In *International Provenance and Annotation Workshop (IPAW '10) (Lecture Notes in Computer Science)*, Vol. 6378. Troy, NY, 148–164. DOI: http://dx.doi.org/10.1007/978-3-642-17819-1_18

David Beckett, Tim Berners-Lee, Eric Prud'hommeaux, and Gavin Carothers. 2014. *Terse RDF Triple Language*. W3C Working Group Recommendation REC-turtle-20140225. World Wide Web Consortium. https://www.w3.org/TR/2014/REC-turtle-20140225/

John Boyer and Glenn Marcy. 2008. *Canonical XML Version 1.1*. W3C Recommendation May 2008. World Wide Web Consortium. http://www.w3.org/TR/2008/REC-xml-c14n11-20080502/

Dan Brickley and R.V. Guha. 2014. *RDF Vocabulary Description Language 1.0: RDF Schema*. W3C Recommendation. World Wide Web Consortium. http://www.w3.org/TR/rdf-schema/

Jeremy J. Carroll. 2002. Matching RDF Graphs. In *First International Semantic Web Conference (ISWC'02), Sardinia, Italy, June 9–12, 2002*. Springer Berlin Heidelberg, Berlin, Heidelberg, 5–15. DOI: http://dx.doi.org/10.1007/3-540-48005-6_3

Jeremy J. Carroll. 2003. Signing RDF Graphs. In *Second International Semantic Web Conference (ISWC'03), Sanibel Island, FL, USA, October 20-23, 2003*. Springer Berlin Heidelberg, 369–384. DOI: http://dx.doi.org/10.1007/978-3-540-39718-2_24

James Cheney. 2013. *Semantics of the PROV Data Model*. W3C Working Group Note NOTE-prov-sem-20130430. World Wide Web Consortium. http://www.w3.org/TR/2013/NOTE-prov-sem-20130430/

James Cheney, Laura Chiticarius, and Wang-Chiew Tan. 2009. Provenance in Databases: Why, How, and Where. *Foundations and Trends in Databases* 1, 4 (2009), 379–474. DOI: http://dx.doi.org/10.1561/1900000006

James Cheney, Paolo Missier, Luc Moreau (eds.), and Tom De Nies. 2013. *Constraints of the PROV Data Model*. W3C Recommendation. World Wide Web Consortium. http://www.w3.org/TR/2013/REC-prov-constraints-20130430/

D. Eastlake and P. Jones. 2001. *US Secure Hash Algorithm 1 (SHA1)*. Technical Report. Internet Engineering Task Force.

D. Eastlake, J. Reagle, D. Solo, Hirsch F., and Roessler T. 2008. *XML-Signature Syntax and Processing (second edition)*. W3C Recommendation. World Wide Web Consortium. http://www.w3.org/TR/xmldsig-core

L. M. R. Gadelha, Jr. and M. Mattoso. 2008. Kairos: An Architecture for Securing Authorship and Temporal Information of Provenance Data in Grid-Enabled Workflow Management Systems. In *IEEE Fourth International Conference on eScience (eScience'08)*. 597–602. DOI:http://dx.doi.org/10.1109/eScience.2008.161

Paul Groth and Luc Moreau (eds.). 2013. *PROV-Overview. An Overview of the PROV Family of Documents*. Technical Report. World Wide Web Consortium. http://www.w3.org/TR/2013/NOTE-prov-overview-20130430/

Ragib Hasan, Radu Sion, and Marianne Winslett. 2009. The Case of the Fake Picasso: Preventing History Forgery with Secure Provenance. In *Proceedings of the 7th Conference on File and Storage Technologies (FAST '09)*. San Francisco, California, 1–14. https://www.usenix.org/conference/fast-09/case-fake-picasso-preventing-history-forgery-secure-provenance

Patrick J. Hayes and Peter F. Patel-Schneider. 2014. *RDF 1.1 Semantics*. W3C Recommendation February 2014. World Wide Web Consortium. https://www.w3.org/TR/2014/REC-rdf11-mt-20140225/

Hook Hua, Curt Tilmes, Stephan Zednik (eds.), and Luc Moreau. 2013. *PROV-XML: The PROV XML Schema*. W3C Working Group Note NOTE-prov-xml-20130430. World Wide Web Consortium. http://www.w3.org/TR/2013/NOTE-prov-xml-20130430/

Trung Dong Huynh, Michael O. Jewell, Amir Sezavar Keshavarz, Danius T. Michaelides, Huanjia Yang, and Luc Moreau. 2013. *The PROV-JSON Serialization*. Member Submission. World Wide Web Consortium. http://www.w3.org/Submission/prov-json/

Trung Dong Huynh and Luc Moreau. 2014. ProvStore: a public provenance repository. In *5th International Provenance and Annotation Workshop (IPAW'14) (Lecture Notes in Computer Science)*. Springer Berlin Heidelberg, Cologne, Germany, 275–277. DOI:http://dx.doi.org/10.1007/978-3-319-16462-5_32

Natalia Kwasnikowska, Luc Moreau, and Jan Van den Bussche. 2015. A Formal Account of the Open Provenance Model. *ACM Trans Web* 9 (February 2015), 44. Issue 2. http://eprints.soton.ac.uk/id/eprint/374183

Timothy Lebo, Satya Sahoo, Deborah McGuinness (eds.), Khalid Behajjame, James Cheney, David Corsar, Daniel Garijo, Stian Soiland-Reyes, Stephan Zednik, and Jun Zhao. 2013. *PROV-O: The PROV Ontology*. W3C Recommendation. World Wide Web Consortium. http://www.w3.org/TR/2013/REC-prov-o-20130430/

Dave Longley and Manu Sporny. 2016. *RDF Dataset Normalization*. Technical Report. World Wide Web Consortium. http://json-ld.github.io/normalization/spec/

Brendan D. McKay and Adolfo Piperno. 2014. Practical graph isomorphism, II. *Journal of Symbolic Computation* 60 (2014), 94 – 112. DOI:http://dx.doi.org/10.1016/j.jsc.2013.09.003

Luc Moreau. 2016. Directed Qualified Pattern, Influence, Non-Influence Relations, Optional Attributes. PROV: Three Years Later workshop. (2016). http://provenanceweek.org/2016/p3yl/papers/paper_87.pdf

Luc Moreau, Ben Clifford, Juliana Freire, Joe Futrelle, Yolanda Gil, Paul Groth, Natalia Kwasnikowska, Simon Miles, Paolo Missier, Jim Myers, Beth Plale, Yogesh Simmhan, Eric Stephan, and Jan Van den Bussche. 2011. The Open Provenance Model core specification (v1.1). *Future Generation Computer Systems* 27, 6 (June 2011), 743–756. DOI:http://dx.doi.org/10.1016/j.future.2010.07.005

Luc Moreau, Trung Dong Huynh, and Danius Michaelides. 2014. An Online Validator for Provenance: Algorithmic Design, Testing, and API. In *17th International Conference on Fundamental Approaches to Software Engineering (FASE'14) (Lecture Notes in Computer Science)*, Vol. 8411. Springer-Verlag, Lyon, France, 291–305. DOI:http://dx.doi.org/10.1007/978-3-642-54804-8_20

Luc Moreau and Bertram Ludaescher. 2008. The First Provenance Challenge. *Concurrency and Computation: Practice and Experience* 20, 5 (April 2008), 409–418. DOI:http://dx.doi.org/10.1002/cpe.1233

Luc Moreau and Paolo Missier (eds.). 2013. *PROV-DM: The PROV Data Model*. W3C Recommendation. World Wide Web Consortium. http://www.w3.org/TR/2013/REC-prov-dm-20130430/

Luc Moreau, Paolo Missier (eds.), James Cheney, and Stian Soiland-Reyes. 2013. *PROV-N: The Provenance Notation*. W3C Recommendation. World Wide Web Consortium. http://www.w3.org/TR/2013/REC-prov-n-20130430/

Martin Odersky, Lex Spoon, and Bill Venners. 2010. *Programming In Scala*. Artima.

ProvRDF 2013. ProvRDF. (May 2013). https://www.w3.org/2011/prov/wiki/ProvRDF

Sarvapali Ramchurn, Edwin Simpson, Joel Fischer, Trung Huynh, Y. Ikuno, Steven Reece, Wenchao Jiang, Feng Wu, Jack Flann, S.J. Roberts, Luc Moreau, T. Rodden, and N.R. Jennings. 2015. HAC-ER: A disaster response system based on human-agent collectives. In *14th International Conference on Autonomous Agents and Multi-Agent Systems*. Istambul, Turkey, 533–541. http://eprints.soton.ac.uk/374070/

Ronald C. Read and Derek G. Corneil. 1977. The graph isomorphism disease. *Journal of Graph Theory* 1, 4 (1 1977), 339–363. DOI:http://dx.doi.org/10.1002/jgt.3190010410

Joseph Reagle. 2008. *XML-Signature Requirements*. W3C Working Draft Oct 1999. World Wide Web Consortium. https://www.w3.org/TR/xmldsig-requirements

Oshani Seneviratne and Ken Beckett. 2016. PROV: Three Years Later workshop. (2016). http://provenanceweek.org/2016/p3yl/papers/paper_81.pdf

Manu Sporny. 2015. *Secure Messaging 1.0*. Technical Report. W3C Web Payments Community Group.

Henk C. A. van Tilborg (Ed.). 2005. *Encyclopedia of Cryptography and Security*. Springer, New York. DOI:http://dx.doi.org/10.1007/0-387-23483-7_364

X. Wang, K. Zeng, K. Govindan, and P. Mohapatra. 2012. Chaining for securing data provenance in distributed information networks. In *IEEE Military Communications Conference (MILCOM'12)*. 1–6. DOI:http://dx.doi.org/10.1109/MILCOM.2012.6415609